ORACLE
PEOPLESOFT ENTERPRISE

Enterprise PeopleTools 8.51 PeopleBook: PeopleCode API Reference

October 2011

ORACLE

Enterprise PeopleTools 8.51 PeopleBook: PeopleCode API Reference
SKU pt8.51tpcr-b1011

Contents

Preface

PeopleCode API Reference Preface ..... **cxvii**

PeopleCode API Reference cxvii
PeopleCode Typographical Conventions cxvii
PeopleBooks and the PeopleSoft Online Library cxviii

Chapter 1

AESection Class **1**

Understanding AESection Class 1
How an AESection is Accessed 2
AESection Example 3
Data Type of an AESection Object 5
Scope of an AESection Object 5
AESection Class Built-in Function 5
AESection Class Methods 5
  AddStep 5
  Close 6
  Open 7
  Save 8
  SetSQL 9
  SetTemplate 10
AESection Class Property 11
  IsOpen 11

Chapter 2

Analytic Calculation Engine Classes **13**

Understanding the Analytic Calculation Engine Classes 13
  Using the Analytic Calculation Engine Classes with Application Engine 14
Running Synchronously 14
Using Trees 14
Error Handling 15
Data Types for Analytic Calculation Engine Classes 16
Scope of Analytic Calculation Engine Classes 16

Analytic Calculation Engine Classes Built-in Functions 16
AnalyticInstance Class Methods 16
CheckAsyncStatus 16
CheckStatus 17
Copy 18
Delete 19
GetAnalyticModel 20
Load 21
RunAsync 22
RunSync 23
Terminate 24
Unload 24
AnalyticInstance Class Properties 25
AnalyticType 25
ID 25
Messages 26
AnalyticModel Class Methods 26
AddMember 26
AttachTree 27
CalculateCube 28
DetachTree 29
GetCubeCollection 30
GetCellProperties 30
GetMembers 31
GetTree 33
Recalculate 34
RenameMember 35
AnalyticModel Class Property 35
Messages 36
CubeCollection Class 36
CubeCollection Class Methods 36
CollapseNode 36
DrillIntoNode 37
DrillOutOfNode 38
ExpandNode 39
GetData 40
GetDimFilter 41
GetDimSort 42
GetLayout 43
GetRowCount 44
GetSlice 44
SetData 45
SetDimensionOrder 46
SetDimFilter 46
SetDimSort 47

SetSlice 48
ShowHierarchy 49
UnsetDimFilter 50
UnsetDimSort 51
CubeCollection Class Property 51
Messages 52

Chapter 3

Analytic Calculation Engine Metadata Classes 53

Understanding the PeopleSoft Analytic Calculation Engine Metadata Classes 53
Using the Analytic Calculation Engine Metadata Classes 54
Inserting and Deleting Objects 54
Building a Rule as an Object 54
Error Handling 55
Data Types of the Analytic Calculation Engine Metadata Objects 55
Scope of Analytic Calculation Engine Metadata Objects 56
How to Import the Analytic Calculation Engine Metadata Classes 56
How to Create an Analytic Calculation Engine Metadata Class Object 57
Analytic Calculation Engine Metadata Classes Constructor 57
AnalyticModelDefn 58
AnalyticModelDefn Class 58
AnalyticModelDefn Class Methods 59
AddCube 59
AddCubeCollection 59
AddDimension 60
AddExplicitDimensionSet 61
AddOrganizer 62
AddUserFunction 62
CopyCube 63
CopyCubeCollection 64
CopyDimension 65
CopyExplicitDimensionSet 65
CopyTo 66
CopyUserFunction 67
Create 68
Delete 68
DeleteCube 69
DeleteCubeCollection 70
DeleteDimension 71
DeleteExplicitDimensionSet 71
DeleteOrganizer 72
DeleteUserFunction 73

Get ... 74
GetCube ... 74
GetCubeCollection ... 75
GetCubeCollectionNames ... 76
GetCubeNames ... 77
GetDimension ... 77
GetDimensionNames ... 78
GetExplicitDimensionSet ... 79
GetExplicitDimensionSetNames ... 80
GetOrganizer ... 80
GetOrganizerNames ... 81
GetUserFunction ... 82
GetUserFunctionNames ... 82
Rename ... 83
RenameCube ... 84
RenameCubeCollection ... 85
RenameDimension ... 85
RenameExplicitDimensionSet ... 86
RenameOrganizer ... 87
RenameUserFunction ... 88
Save ... 89
Validate ... 89
AnalyticModelDefn Properties ... 90
CircularFormulaWarn ... 90
Description ... 91
IsValid ... 91
LongDescription ... 91
MaxDelta ... 91
MaxIterations ... 92
Messages ... 92
Name ... 93
ResolveCircularDeps ... 93
DimensionDefn Class ... 93
DimensionDefn Class Properties ... 93
AggregateSequence ... 94
AggregationUserFunction ... 94
Comments ... 94
Name ... 94
TotalMemberName ... 95
ExplicitDimensionSet Class ... 95
ExplicitDimensionSet Methods ... 95
AttachDimension ... 95
DetachDimension ... 96
GetDimensionNames ... 97
ExplicitDimensionSet Properties ... 97

Name 97
SequenceNumber 98
CubeDefn Class 98
CubeDefn Class Methods 98
AttachDimension 98
DetachDimension 99
GetCauses 100
GetCircularDeps 101
GetDimensionAggregate 101
GetEffects 102
GetDimensionNames 103
GetRule 103
SetDimensionAggregate 104
SetRule 105
UsesDimension 105
CubeDefn Class Properties 106
CalcAggregates 106
Comments 106
DimensionCount 107
FormatType 107
IsVirtual 107
Name 107
ValueDimensionName 108
CubeCollectionDefn Class 108
CubeCollectionDefn Class Methods 108
AttachCube 108
DetachCube 109
GetAggregateMapping 110
GetCubeNames 111
GetDimensionNames 111
GetDimSort 112
GetFieldMapping 113
GetFilter 114
GetPersistAggregate 114
SetAggregateMapping 115
SetDimSort 116
SetFieldMapping 117
SetFilter 118
SetPersistAggregate 119
UsesCube 120
UsesDimension 121
CubeCollectionDefn Class Properties 121
AggregateRecName 122
Comments 122
CubeCount 122

DimensionCount 122
Name 122
RecordName 123
UserFunctionDefn Class 123
UserFunctionDefn Class Methods 123
GetRule 123
SetRule 124
UserFunctionDefn Class Properties 124
Comments 124
Name 125
OrganizerDefn Class 125
OrganizerDefn Class Methods 125
AttachPart 125
DetachPart 126
GetPartNames 127
UsesPart 128
OrganizerDefn Class Properties 129
Comments 129
Name 129
RuleDefn Class 129
RuleDefn Class Methods 129
AddRuleExpression 129
GenerateRule 130
RuleDefn Class Property 131
RuleString 131
RuleExpressions Classes 131
Using the Constants Class 132
Assignment Class 132
Assignment Class Method 132
GenerateRule 132
Assignment Class Properties 133
Expression 133
Variable 133
Comparison Class 133
Comparison Class Method 134
GenerateRule 134
Comparison Class Properties 135
Operand1 135
Operand2 135
Type 136
Constant Class 136
Constant Class Method 137
GenerateRule 137
Constant Class Properties 138
Type 138

- Value ... 138
- Constants Class ... 139
- Cube Class ... 139
- Cube Class Methods ... 139
  - AddIndex ... 139
  - GenerateRule ... 140
  - GetIndexes ... 140
- Cube Class Property ... 141
  - Name ... 141
- ExpressionBlock Class ... 141
- ExpressionBlock Methods ... 142
  - AddRuleExpression ... 142
  - GetRuleExpressions ... 142
- FunctionCall Class ... 143
- FunctionCall Class Methods ... 147
  - AddArgument ... 147
  - GenerateRule ... 147
  - GetArguments ... 148
- FunctionCall Class Properties ... 149
  - Name ... 149
  - Type ... 149
- MemberReference Class ... 149
- MemberReference Class Method ... 149
  - GenerateRule ... 149
- MemberReference Class Properties ... 150
  - Dimension ... 150
  - Member ... 150
- Operation Class ... 150
- Operation Class Method ... 151
  - GenerateRule ... 151
- Operation Class Properties ... 152
  - Operand1 ... 152
  - Operand2 ... 152
  - Type ... 152
- Variable Class ... 153
- Variable Class Method ... 153
  - GenerateRule ... 154
- Variable Class Properties ... 154
  - Name ... 154
  - Type ... 154
- Analytic Model Metadata Classes Examples ... 155
  - Creating an Analytic Model Code Example ... 155

Chapter 4

Analytic Grid Classes **159**

Understanding the Analytic Calculation Engine Classes 159
Using the Analytic Grid in PeopleCode 160
Using Freeze Column Mode 161
Error Handling 162
Data Types of the Analytic Grid Objects 162
Scope of Analytic Grid Objects 162
AnalyticGrid Class Reference 162
AnalyticGrid Class Built-in Function 163
AnalyticGrid Class Methods 163
GetColumn 163
GetCubeCollection 164
LoadData 165
SetAnalyticInstance 165
SetLayout 166
AnalyticGrid Class Properties 167
Inactive 167
Label 167
ShowGridLines 167
SlicerVisible 168
SummaryText 168

Chapter 5

Analytic Type Classes **169**

Understanding the Analytic Calculation Engine Classes 169
Using the Analytic Type Classes 170
Error Handling 170
Data Types of the Analytic Type Objects 170
Scope of Analytic Type Objects 171
How to Import the Analytic Type Classes 171
How to Create an Analytic Type Class Object 171
Analytic Type Classes Constructor 172
AnalyticTypeDefn 172
AnalyticTypeDefn Class Methods 173
AddModel 173
AddRecord 174
Create 174
CopyTo 175

Delete 176
DeleteModel 176
DeleteRecord 177
Get 178
GetModel 178
GetModelNames 179
GetRecord 180
GetRecordNames 180
Rename 181
Save 182
AnalyticTypeDefn Class Properties 182
AppClassPath 182
Comments 182
Description 183
Name 183
OwnerID 183
AnalyticTypeModelDefn Class Properties 183
Name 183
Type 184
AnalyticTypeRecordDefn Class Methods 184
GetSelectedField 184
SetSelectedField 185
UnsetSelectedField 185
AnalyticTypeRecordDefn Class Properties 186
Callback 186
Description 186
Name 186
Readable 187
ReadOnce 187
ScenarioManaged 187
SyncOrder 187
Writeable 187
AnalyticType Classes Example 188

Chapter 6

Application Classes 189

Understanding Application Classes 189
When Would You Use Application Classes? 190
Application Classes General Structure 190
Class Name 191
Class Extension 192
Declaration of Public External Interface 193

- Access Control and the Declaration of Protected Properties and Methods ... 194
- Declaration of Private Instance Variables and Methods ... 198
- Definition of Methods ... 199
- Declaration of Abstract Methods and Properties ... 202
- Interfaces ... 203
- Constructors ... 204
- External Function Declarations ... 206

Naming Standards ... 208
- Naming Packages ... 208
- Naming Classes ... 208
- Naming Methods ... 208
- Naming Properties ... 209

Import Declarations ... 209

Self-Reference ... 210
- Using %This with Constructors ... 211

Properties and Constants ... 212
- Differentiating Between Properties and Methods ... 213
- Overriding Properties ... 214
- Using Get and Set with Properties ... 215
- Using Read-Only Properties ... 215
- Using Methods and Properties for Collections ... 216
- Declaring Constants ... 216

Using Variables in Application Classes ... 217
- Placement of Variable Declarations ... 217
- Declaring Private Instance Variables ... 217
- Global Variables ... 217
- Overriding Variables and Properties ... 218

Superclass Reference ... 218

Downcasting ... 219

Exception Handling ... 220

Commenting and Documenting Application Classes ... 221
- Understanding Comments and Documentation ... 221
- Method Header Comments ... 221
- Class Header Comments ... 222

Designing Base Classes ... 223
- Base Classes ... 223
- Abstract Base Classes ... 223
- Generic Base Classes ... 223
- Utility Classes ... 224

Declaration of Application Classes ... 224
- Declaring Application Classes ... 224
- Importing Class Names ... 224
- Referencing Superclasses ... 225
- Declaring Private Methods ... 225

Scope of an Application Class ... 225

Application Classes Built-in Functions and Language Constructs 225
Class 225
Get 227
Import 228
Interface 229
Method 230
Set 231

Chapter 7

API Repository 233

Understanding the PeopleSoft API Repository 233
The PeopleSoft API Repository 234
Example of Accessing the Repository Using PeopleCode 235
Example of Accessing the Repository by Using Visual Basic 239
Repository Properties 241
Bindings 241
Namespaces 241
Bindings Collection Properties 242
Count 242
Bindings Collection Methods 242
Item 242
Bindings Properties 243
Name 243
Bindings Methods 243
Generate 243
Namespaces Collection Properties 244
Count 244
Namespaces Collection Methods 244
Item 244
ItemByName 245
Namespaces Properties 246
Classes 246
Name 246
Namespaces Methods 246
CreateObject 246
ClassInfo Collection Properties 247
Count 247
ClassInfo Collection Methods 247
Item 247
ItemByName 248
ClassInfo Properties 249
Documentation 249

Methods 249
Name 249
Properties 249
MethodInfo Collection Properties 250
Count 250
MethodInfo Collection Methods 250
Item 250
ItemByName 251
MethodInfo Properties 252
Arguments 252
Documentation 252
Name 252
Type 252
PropertyInfo Collection Properties 253
Count 253
PropertyInfo Collection Methods 253
Item 254
ItemByName 254
PropertyInfo Properties 255
Documentation 255
Name 255
Type 255
Usage 256

Chapter 8

Array Class 257

Understanding Arrays 257
Creating Arrays 257
Populating an Array 259
Removing Items From an Array 260
Creating Empty Arrays 260
Creating and Populating Multi-Dimensional Arrays 261
Using Flattening and Promotion 264
Declaring Array Objects 264
Understanding the Scope of an Array Object 265
Array Class Built-in Functions 265
Array Class Methods 265
Clone 265
Find 266
Get 267
Join 268
Next 271

Pop 272
Push 273
Replace 275
Reverse 277
Set 278
Shift 278
Sort 279
Subarray 282
Substitute 282
Unshift 284
Array Class Properties 284
Dimension 285
Len 285

Chapter 9

BPEL Classes 287

Understanding the BPEL Classes 287
Scope of the BPEL Classes 287
Data Types of the BPEL Classes 288
How to Import the BPEL Classes 288
How to Create a BPEL Object 288
BPEL Classes Constructors 289
AsyncFFSend 289
BPELUtil 290
IBUtil 290
AsyncFFSend Class 291
AsyncFFSend Class Method 291
OnRequestSend 291
BPELUtil Class 292
BPELUtil Class Methods 292
GetASyncBPELProcessInstanceUrl 292
GetBPELProcessBrowserUrl 293
GetOperationType 293
GetSyncBPELProcessInstanceUrl 294
LaunchASyncBPELProcess 295
LaunchSyncBPELProcess 296
UpdateConnectorResponseProperties 297
IBUtil Class 298
IBUtil Class Methods 298
GetBPELConsoleUrl 298
GetBPELDomain 299
GetMessageId 300

GetNode 300
GetNumberOfRoutings 301
GetStatus 301
GetTransactionId 302

Chapter 10

Business Interlink Class 303

Understanding Business Interlink Class 303
Using the Interlink Object 304
Deciding Which Methods to Use 304
Executing the Business Interlink Object 305
Supporting Batch Input and Output 305
Supporting Rowsets 305
Using the Flat Table Methods 305
Supporting Dynamic Output 305
Using Hierarchical Data (BIDocs) 306
Using the Incoming Business Interlink Methods and Properties 309
Understanding the State of an Interlink Object 310
Using Business Interlink with Application Engine 310
Using the Data Type of a Business Interlink Object 311
Understanding the Scope of a Business Interlink Object 311
Business Interlink Class Built-in Functions 311
Business Interlink Class Methods 311
AddDoc 311
AddInputRow 313
AddNextDoc 314
AddValue 317
BulkExecute 319
Clear 324
Execute 325
FetchIntoRecord 327
FetchIntoRowset 329
FetchNextRow 332
GetCount 334
GetDoc 335
GetFieldCount 338
GetFieldType 339
GetFieldValue 340
GetInputDocs 341
GetNextDoc 342
GetOutputDocs 344
GetPreviousDoc 345

GetStatus 347
GetValue 349
InputRowset 350
MoveFirst 352
MoveNext 353
MoveToDoc 354
ResetCursor 356
Incoming Business Interlink Methods 357
AddAttribute 357
AddComment 358
AddProcessInstruction 359
AddText 361
CreateElement 362
GenXMLString 363
GetAttributeName 364
GetAttributeValue 365
GetNode 366
ParseXMLString 367
Incoming Business Interlink Properties 368
AttributeCount 368
ChildNodeCount 369
NodeName 370
NodeType 370
NodeValue 371
Business Interlink Class Property 372
StopAtError 372
Configuration Parameters 372
Interlink Plug-in Configuration Parameters 373
URL Configuration Parameter 373
BIDocValidate Configuration Parameter 374

Chapter 11

Charting Classes 375

Understanding the Charting Classes 375
Creating PeopleSoft Charts 378
Creating a Chart on a Page 379
Creating a Chart Using an iScript 382
Font Considerations 382
Component Processor Considerations 383
Translation Considerations 383
Label Considerations 383
Style Sheets 383

Using Style Sheets with RevertToPre850 384
Using Style Sheets with PeopleTools Version 8.50 Charts 386
Using the Chart Class 394
Chart Terms 395
Creating Charts Using the Chart Class 395
Chart Class Chart Types 396
Data Type of a Chart Object 397
Scope of a Chart Object 397
Error Handling 397
Chart Class Methods and Properties by Category 397
Using the Gantt Class 399
Understanding Gantt Chart Terms 400
Using Gantt Charts in PeopleSoft Pure Internet Architecture 400
Creating Gantt Charts Using the Gantt Class 403
Specifying DateTime Axis Formats 404
Working with Start and End Dates 405
Using Gantt Glyphs 405
Using Style Sheets with Gantt Charts 406
Data Type of a Gantt Object 406
Scope of a Gantt Object 406
Error Handling 406
Using the OrgChart Class 407
Understanding Organization Chart Terms 407
Using Organization Charts in PeopleSoft Pure Internet Architecture 409
Creating Organization Charts Using the OrgChart Class 411
Specifying Scroll Type 415
Organization Chart Actions and Events 416
Organization Chart Subrecord Definitions 417
Minimum Methods 424
Data Type of an OrgChart Object 425
Scope of an OrgChart Object 425
Using the RatingBoxChart Class 425
Understanding Rating Box Chart Terms 426
Using Rating Box Charts in PeopleSoft Pure Internet Architecture 427
Creating Rating Box Charts Using the RatingBoxChart Class 428
Rating Box Chart Subrecord Definition 429
Minimum Methods 430
Minimum Properties 430
Data Type of a RatingBoxChart Object 430
Scope of a RatingBoxChart Object 430
Charting Classes Reference 431
Charting Classes Built-in Functions 431
Chart Class Methods 432
Refresh 432
Reset 433

SetColorArray 433
SetData 435
SetDataAnnotations 437
SetDataColor 438
SetDataGlyphScale 438
SetDataHints 439
SetDataSeries 443
SetDataURLs 443
SetDataXAxis 444
SetDataYAxis 446
SetLegend 446
SetOLData 447
SetOLDataAnnotations 448
SetOLDataGlyphScale 449
SetOLDataSeries 450
SetOLDataXAxis 450
SetOLDataYAxis 451
SetXAxisLabels 452
SetYAxisLabels 452
Chart Class Properties 453
DataStartRow 454
DataWidth 454
GridLines 455
GridLineType 455
HasLegend 456
Height 456
ImageMap 457
IsDrillable 457
IsPlainImage 458
IsTrueXY 458
IsYAxisInteger 458
LegendMaxEntries 459
LegendPosition 459
LegendStyle 460
LineType 460
MainTitle 461
MainTitleOrient 461
MainTitleStyle 461
OLLineType 462
OLType 462
RevertToPre850 463
RotationAngle 463
ShowCrossHair 464
Style 464
StyleSheet 464

Type 465
Width 466
XAxisCross 466
XAxisCrossPoint 467
XAxisLabelOrient 468
XAxisMax 468
XAxisMin 468
XAxisPrecision 469
XAxisScaleResolution 469
XAxisStyle 470
XAxisTicks 470
XAxisTitle 470
XAxisTitleOrient 471
XAxisTitleStyle 471
XRotationAngle 471
YAxisCrossPoint 472
YAxisLabelOrient 472
YAxisMax 473
YAxisMin 473
YAxisPrecision 473
YAxisScaleResolution 474
YAxisStyle 474
YAxisTicks 475
YAxisTitle 475
YAxisTitleOrient 475
YAxisTitleStyle 476
YRotationAngle 476
ZRotationAngle 476
Gantt Class Methods 477
Refresh 477
Reset 478
SetActualEndDate 478
SetActualStartDate 479
SetActualTaskBarColor 480
SetChartArea 480
SetDayFormat 481
SetHourFormat 482
SetLegend 483
SetMinuteFormat 484
SetMonthFormat 485
SetPlannedEndDate 486
SetPlannedStartDate 486
SetPlannedTaskBarColor 487
SetSecondFormat 488
SetTableXScrollbar 489

SetTaskAppData 489
SetTaskAppDataTitles 490
SetTaskBarURL 491
SetTaskData 492
SetTaskDependencyChildID 492
SetTaskDependencyData 493
SetTaskDependencyParentID 494
SetTaskDependencyType 494
SetTaskDependencyURL 496
SetTaskDrill 497
SetTaskExpanded 497
SetTaskHints 498
SetTaskID 499
SetTaskLabel 499
SetTaskLevel 500
SetTaskMilestone 501
SetTaskName 502
SetTaskProgress 503
SetTaskProgressBarColor 503
SetWBSNumbering 504
SetYearFormat 505
Gantt Class Properties 506
AxisEndDateTime 506
AxisStartDateTime 506
DataEndDateTime 507
DataStartDateTime 507
DataStartRow 507
DataWidth 507
GridLines 508
GridLineType 509
HasLegend 509
Height 510
ImageMap 510
InteractiveEnd 510
InteractiveMove 511
InteractiveProgress 511
InteractiveStart 511
IsDrillable 511
IsPlainImage 512
LegendPosition 513
LegendStyle 513
MainTitle 514
MainTitleStyle 514
PixelsPerRow 514
RevertToPre850 515

ShowTaskLabels 515
Style 516
StyleSheet 516
TaskDependencyLineType 516
TaskMilestoneGlyph 517
TaskTitle 517
Width 518
XAxisPosition 518
YAxisPosition 518
OrgChart Class Methods 519
SetCrumbData 519
SetCrumbRecord 520
SetDropdownData 520
SetDropdownRecord 521
SetLegend 521
SetLegendImg 522
SetNodeData 523
SetNodeRecord 524
SetPopUpNodeData 524
SetPopUpNodeRecord 525
OrgChart Class Properties 525
CenterFocusNode 526
ChartScrollType 526
Collapsed_Msg 527
CollapsedImage 527
CollapseMainIconSpace 527
CrumbDescrStyle 528
CrumbMaxDisplayLength 528
CrumbSeparatorImage 528
DefaultImage 529
Direction 529
DropDownBoxStyle 529
Expanded_Msg 530
ExpandedImage 530
HasLegend 530
Height 530
ImageHeight 531
ImageLocation 531
ImageMouseoverMagnificationFactor 532
LegendPosition 532
LegendStyle 532
LegendTopSpace 533
MainTitle 533
MainTitleStyle 533
MaxDropdownDisplayItem 533

MaxPopupDisplayNode ... 534
NodeDescr1Style ... 534
NodeDescr2Style ... 534
NodeDescr3Style ... 535
NodeDescr4Style ... 535
NodeDescr5Style ... 535
NodeDescr6Style ... 536
NodeDescr7Style ... 536
NodeMaxDisplayDescLength ... 536
OptimizeHorizontalSpace ... 537
OptimizeVerticalSpace ... 537
PopupHeaderStyle ... 538
PopupNodeDescr1Style ... 538
PopupNodeDescr2Style ... 538
PopupNodeDescr3Style ... 539
PopupNodeDescr4Style ... 539
PopupNodeDescr5Style ... 539
PopupNodeDescr6Style ... 540
PopupNodeDescr7Style ... 540
PopupNodeDescr8Style ... 540
Style ... 541
SuppressPeopleCodeOnNode ... 541
SuppressPeopleCodeOnImage ... 541
UnlinkCrumbDescrStyle ... 542
VerticalSpace ... 542
Width ... 542
RatingBoxChart Class Methods ... 543
SetLegend ... 543
SetLegendImg ... 544
SetRBNodeData ... 544
SetRBNodeRecord ... 545
SetXAxisLabels ... 545
SetYAxisLabels ... 546
RatingBoxChart Class Properties ... 547
BoxMaxDisplayItems ... 547
DraggedNodeStyle ... 547
GridLineType ... 548
HasLegend ... 548
Height ... 548
IconOnlySelectedQuadrantStyle ... 549
IsDragable ... 549
LegendPosition ... 549
MainTitle ... 550
MainTitleStyle ... 550
NDMaxDisplayDescLength ... 550

PopUpHeaderStyle 551
PopUpHeight 551
PopUpStyle 552
PopUpWidth 552
SelectedQuadrantStyle 552
ShowNodeDescription 553
Style 553
ViewAllStyle 553
Width 554
XAxisBoxNum 554
XAxisLabelStyle 554
XAxisTitle 555
XAxisTitleStyle 555
YAxisBoxNum 555
YAxisLabelStyle 556
YAxisTitle 556
YAxisTitleStyle 556
Charting Examples 557
Creating a Chart Using the Chart Class 557
Creating a Gantt Chart 585
Creating an Organization Chart 588
Creating a Rating Box Chart 592
Creating a Chart Using an iScript 596

Chapter 12

Component Interface Classes 599

Understanding Component Interface Class 599
Life Cycle of a Component Interface 600
Setting Component Interface Keys 602
Standard and User-Defined Component Interface Methods 603
Accessing Component Interface Standard Properties 604
Accessing User-Defined Component Interface Properties 605
Data Type of a Component Interface Object 606
Scope of a Component Interface Object 606
Implementing a Component Interface 607
Component Interface Methods and Timeouts 607
Traversing a Component Interface and Using Data Collections 608
Working With Effective-Dated Data 609
How a Developer Uses GetEffectiveItem 610
Reusing Existing Code 611
Differences in Search Dialog Processing 611
Differences in PeopleCode Event and Function Behavior 611

Using %Menu Conditions 612
Session Class Methods 612
FindCompIntfcs 612
GetCompIntfc 613
Component Interface Collection Methods 613
First 614
Item 614
Next 614
Component Interface Collection Property 615
Count 615
Component Interface Class Methods 615
Cancel 615
CopyRowset 616
CopyRowsetDelta 618
CopySetupRowset 620
CopySetupRowsetDelta 623
Create 626
Find 627
Get 627
GetPropertyByName 628
Save 630
SetPropertyByName 631
Component Interface Class Properties 632
ComponentName 632
CreateKeyInfoCollection 632
EditHistoryItems 632
FindKeyInfoCollection 633
GetDummyRows 633
GetHistoryItems 634
GetKeyInfoCollection 634
InteractiveMode 635
PropertyInfoCollection 635
StopOnFirstError 635
Data Collection Methods 636
CurrentItem 636
DeleteItem 636
GetEffectiveItem 638
GetEffectiveItemNum 639
InsertItem 640
Item 641
ItemByKeys 641
Data Collection Properties 642
Count 643
CurrentItemNumber 643
Data Item Class Property 643

ItemNum 643
Accessing the Structure of a Component Interface 644
CompIntfPropInfoCollection Collection 644
CompIntfPropInfoCollection Collection Methods 647
First 647
Item 648
Next 648
CompIntfPropInfoCollection Collection Properties 648
Count 649
CompIntfPropInfoCollection Object Properties 649
Format 649
IsCollection 650
IsReadOnly 650
Key 650
LabelLong 651
LabelShort 651
Length 651
Name 652
Prompt 652
PropertyInfoCollection 652
Required 652
Type 653
Xlat 653
YesNo 654
Component Interface Examples 654
Create a New Instance of Data Example 654
Getting an Existing Instance of Data Example 657
Retrieving a List of Instance of Data Example 659
Inserting Effective-Dated Data Example 661
Inserting Effective-Dated Data Example Using Visual Basic 664
Inserting or Deleting a Row of Data Example 666

Chapter 13

Connected Query Classes 671

Understanding the Connected Query Classes 671
Importing Connected Query Classes 672
CONQRSMGR Class 672
CONQRSMGR Class Constructor 672
CONQRSMGR 672
CONQRSMGR Class Methods 673
CleanOutputFile 673
Close 673

CopyDefn ... 674
DeleteDefn ... 674
ExistsByName ... 675
GetSampleXMLString ... 675
Open ... 676
ResetQueriesPrompt ... 677
Run ... 677
RunToWindow ... 678
RunToXMLFormattedString ... 679
Save ... 680
SaveRunControlParms ... 681
SetRunControlData ... 681
Validate ... 682
ValidateIfFieldsMapped ... 683
ValidateRunControlParms ... 683
CONQRSMGR Class Properties ... 684
Const ... 684
Description ... 684
Details ... 684
ErrString ... 685
ExecutionLog ... 685
IgnoreRelFldOutput ... 685
IsChanged ... 686
IsDebugMode ... 686
IsInit ... 686
IsPublic ... 687
LastUpdatedBy ... 687
LastUpdateDTTM ... 687
MaxRowsPerQuery ... 687
Name ... 687
ObjectRowset ... 688
OprID ... 688
OutProcessFileName ... 688
PropertyArray ... 689
QueriesPromptsArray ... 691
QueryArray ... 692
RegisteredBy ... 692
RegisteredDTTM ... 692
RunCntlId ... 692
RunControlParArray ... 693
RunMode ... 693
SchedInfo ... 695
SchedRequestType ... 696
SecProfile ... 696
ShowFormattedXML ... 696

Status ... 697
XMLDataFileName ... 697
XMLDataFullName ... 697
CONQRSPROPERTY Class ... 698
CONQRSPROPERTY Class Properties ... 698
PROPDEFAULT ... 698
PROPNAME ... 698
PROPVALUE ... 698
PROPVALUEARRAY ... 699
PROPVALUEARRAYDESC ... 699
CONQRS_CONST Class ... 699
CONQRS_CONST Class Properties ... 699
InitExisting ... 699
InitNew ... 700
MsgSet ... 700
RunCntlId_Auto ... 700
RunMode_Prev ... 701
RunMode_Prev_DefRowNumber ... 701
RunMode_Sample ... 702
RunMode_Sched_File ... 702
RunMode_Sched_Web ... 702
RunMode_Window ... 703
RunMode_XMLFormattedString ... 703
SchedRequest_CQR ... 704
SchedRequest_XMLP ... 704
Stat_Active ... 704
Stat_InActive ... 704
Stat_InProgress ... 704
QUERYITEMPROMPT Class ... 705
QUERYITEMPROMPT Class Properties ... 705
QueryName ... 705
QueryPromptRecord ... 705
SCHED_INFO Class ... 706
SCHED_INFO Class Properties ... 706
AE_ID ... 706
DIRLOCATION ... 707
OPRID ... 707
OUTDESTTYPE ... 707
PRCSFILENAME ... 707
PROCESS_INSTANCE ... 708
RUN_CNTL_ID ... 708
SEC_PROFILE Class ... 708
SEC_PROFILE Class Constructor ... 708
SEC_PROFILE ... 709
SEC_PROFILE Class Properties ... 709

CanCreatePublic 709
CanModify 709
CanRunQuery 709
UTILITY Class 710
UTILITY Class Constructor 710
UTILITY 710
UTILITY Class Methods 710
CheckQryForTreePrompt 710
CheckQrySecurity 711
GetQueryScopeByName 712
ValidateObjectID 713
Connected Query Classes Examples 713
Running a Connected Query in Scheduled Mode 713
Running a Connected Query in Preview Mode or Background Mode 715
Iterating Through a Connected Query Opened for Editing 717

Chapter 14

Crypt Class 721

Understanding the Crypt Class 721
Creating a Crypt Object 721
Declaring Crypt Objects 721
Scope of a Crypt Object 722
Crypt Class Methods 722
FirstStep 722
GoToStep 723
LoadLibrary 723
NextStep 724
Open 725
SetParameter 725
UpdateData 726
Crypt Class Properties 727
Result 727
Verified 727

Chapter 15

Document Classes 729

Understanding the Document Classes 729
Data Type and Scope of the Document Classes 729
Document Classes Built-in Functions 730
Document Class 730

Document Class Methods 730
GenXmlString 730
GetDocumentKey 731
GetElement 732
GetRowset 733
GetSchema 734
ParseXmlFromFile 735
ParseXmlFromURL 735
ParseXmlString 736
UpdateFromRowset 737
ValidateData 737
Document Class Properties 738
DocumentElement 738
DocumentKey Class 739
DocumentKey Class Methods 739
SetDocumentKey 739
DocumentKey Class Properties 740
DocumentName 740
PackageName 740
Version 740
Primitive Class 741
Primitive Class Methods 741
GetEnumName 741
GetParent 742
GetPath 742
Primitive Class Properties 743
ElementType 743
EnumCount 743
IsChanged 743
IsInitialized 744
IsRequired 744
MaxDefinedDecimalLength 744
MaxDefinedLength 744
Name 745
OrigValue 745
PrimitiveSubType 745
PrimitiveType 746
SequenceNumber 747
Value 747
Compound Class 748
Compound Class Methods 748
GetParent 748
GetPath 749
GetPropertyByIndex 749
GetPropertyByName 750

GetUniqueKey 751
Compound Class Properties 751
ElementType 751
IsChanged 752
IsInitialized 752
IsRequired 752
Name 752
PropertyCount 752
SequenceNumber 753
Collection Class 753
Collection Class Methods 753
AppendItem 753
CreateItem 754
DeleteItem 755
GetItem 756
GetParent 756
GetPath 757
InsertItem 757
Collection Class Properties 758
CollectionElementType 758
Count 759
DefinedMaxOccurs 759
DefinedMinOccurs 759
ElementType 759
IsChanged 759
IsInitialized 760
IsRequired 760
Name 760
SequenceNumber 760

Chapter 16

Exception Class 761

Understanding Exception Class 761
How Exceptions Are Thrown 762
When to Use Exceptions 762
Try-Catch Sections 764
Data Type for Exception Class Objects 765
Scope of Exception Class Objects 765
Exception Class Built-in Functions 765
Exception Class Methods 765
GetSubstitution 765
Output 766

SetSubstitution 767
ToString 768
Exception Class Properties 768
Context 769
DefaultText 769
MessageNumber 769
MessageSetNumber 769
MessageSeverity 770
StackTrace 770
SubstitutionCount 771

Chapter 17

Feed Classes 773

Understanding the Feed Classes 773
Importing Feed Classes 775
Feed Class 776
Feed Class Constructor 776
Feed 776
Feed Class Methods 776
delete 777
equals 777
execute 778
getAttribute 779
load 780
populatePrefData 781
publishToSites 782
resetFeedAttributes 783
resetFeedSecurities 784
save 785
saveAs 785
setAttribute 786
setDataSourceById 787
unpublishFromSites 788
Feed Class Properties 789
Authorized 789
CategoryID 789
CreateDTTM 790
CreateNode 790
CreateOprID 790
CreatePortal 790
DataSource 790
DataTypeID 791

- Description 791
- FeedAttributes 791
- FeedAuthorizationOprID 791
- FeedAuthorizationOprPWD 792
- FeedAuthorizationType 792
- FeedCacheTime 792
- FeedCacheType 792
- FeedContentUrl 793
- FeedFactory 793
- FeedFormat 793
- FeedSecurities 793
- FeedSecurityType 794
- FeedUrl 794
- HasAdminParams 794
- HasUserParams 794
- IBOperationName 795
- ID 795
- LastPubDTTM 795
- LastUpdDTTM 795
- LastUpdOprID 796
- NameSpaceID 796
- ObjectType 796
- OperatingMode 796
- OwnerID 797
- PublishedInSites 797
- Title 797
- Utility 798

FeedFactory Class 798

FeedFactory Class Constructor 798
- FeedFactory 799

FeedFactory Class Methods 799
- convertFeedLinksToHoverMenu 799
- convertFeedLinksToOPML 800
- createFeed 801
- deleteFeed 802
- genFeedUrl 802
- genUniqueFeedId 803
- getAllPagedFeedLinks 804
- getCategory 805
- getDataSource 806
- getFeed 807
- getFeedDoc 809
- getFeedLink 810
- getFeedLinks 810
- getRelatedFeedsHoverMenu 811

getRelativePageLinkForFeed 812
FeedFactory Class Properties 814
DataSources 814
Feeds 814
Utility 814
DataSource Class 815
DataSource Class Constructor 815
DataSource 815
DataSource Class Methods 816
addParameter 816
equals 817
getAttributeById 818
getContentUrl 819
getDataSecurity 819
getParameterById 820
getParameterDetail 821
getSettingById 821
getSettingDetail 822
isCurrentUserAdmin 823
isCurrentUserAuthorized 823
onDelete 824
onSave 825
resetParameters 825
DataSource Class Properties 826
AdminPersonalizationPage 826
AllowCustomParameters 826
AllowRealTimeFeedSecurity 827
DataSourceType 827
DefaultFeedAttributes 827
DefaultIBOperationName 827
Description 827
HasParameters 828
HasSettings 828
IBOperations 828
ID 828
LongDescription 828
ObjectType 829
Parameters 829
ParametersCompleted 829
Parent 829
PortalSpecificPersonalization 830
Settings 830
SettingsCompleted 830
UserPersonalizationPage 830
Utility 831

DataSourceParameter Class 831
DataSourceParameter Class Constructor 831
    DataSourceParameter 831
DataSourceParameter Class Methods 832
    addUserValue 832
    clone 833
    equals 833
    resetUserValues 834
    setRangeFromFieldTranslates 834
    validateValue 835
DataSourceParameter Class Properties 835
    AllowChangesToRequired 836
    DefaultValue 836
    DefaultValueForDisplay 836
    Description 836
    EditType 836
    EvaluatedValue 837
    FieldType 837
    ID 838
    Name 838
    ObjectType 838
    Parent 838
    PromptTable 839
    Range 839
    Required 839
    SkipValidityChecks 839
    UsageType 840
    UserValues 840
    Utility 840
    Value 841
    ValueForDisplay 841
DataSourceParameterValue Class 841
DataSourceParameterValue Class Constructor 842
    DataSourceParameterValue 842
DataSourceParameterValue Class Methods 842
    clone 842
    equals 843
DataSourceParameterValue Class Properties 843
    Description 844
    ID 844
    Name 844
    ObjectType 844
    OrderNumber 844
    Parent 845
    Value 845

DataSourceSetting Class 845
DataSourceSetting Class Constructor 845
DataSourceSetting 846
DataSourceSetting Class Methods 846
clone 846
equals 847
setRangeFromFieldTranslates 847
DataSourceSetting Class Properties 848
DefaultValue 848
DisplayOnly 848
EditType 848
Enabled 849
FieldType 849
ID 850
LongName 850
Name 850
ObjectType 850
Parent 850
PromptTable 851
Range 851
RefreshOnChange 851
RelatedFieldValue 851
Required 852
ShowRelatedField 852
Utility 852
Value 853
Visible 853
Utility Class 853
Utility Class Constructor 853
Utility 854
Utility Class Methods 854
dateStringToUserPref 854
datetimeToString 855
dateToString 855
decodeXML 856
encodeXML 857
evaluateSysVar 858
genNameSpaceID 859
getExceptionText 859
getFeedDoc 860
getFeedMimeType 861
getFieldTranslates 861
getNodeValue 862
getUserDateFormat 863
getUserDatetimeFormat 863

getUserInfo ... 864
httpStringToDatetime ... 864
join ... 865
join2D ... 866
setMessageHeadersAndMimeType ... 867
setNodeValue ... 868
showException ... 869
showInvalidValueException ... 869
split ... 870
split2D ... 871
stringToDate ... 872
stringToDatetime ... 872
validateSysVar ... 873
viewStringAsAttachment ... 874
Utility Class Properties ... 874
ATTACHMENT_URL ... 875
AUTHTYPE_PERM ... 875
AUTHTYPE_ROLE ... 875
DSPARAMETER_INCREMENTAL ... 875
DSPARAMETER_MAXROW ... 876
DSPARAMETER_SF_MAXMINUTES ... 876
DSPARAMETER_SF_PAGING ... 876
EDITTYPE_NOTABLEEDIT ... 877
EDITTYPE_PROMPTTABLE ... 877
EDITTYPE_TRANSLATETABLE ... 877
EDITTYPE_YESNO ... 877
FEEDATTRIBUTE_AUTHOR ... 878
FEEDATTRIBUTE_CLOUD ... 878
FEEDATTRIBUTE_COMPLETE ... 878
FEEDATTRIBUTE_CONTRIBUTOR ... 878
FEEDATTRIBUTE_COPYRIGHT ... 879
FEEDATTRIBUTE_EXPIRES ... 879
FEEDATTRIBUTE_ICONURL ... 879
FEEDATTRIBUTE_LOGOURL ... 879
FEEDATTRIBUTE_MANAGINGEDITOR ... 880
FEEDATTRIBUTE_MAXAGE ... 880
FEEDATTRIBUTE_PERSINSTRUCTION ... 880
FEEDATTRIBUTE_RATING ... 880
FEEDATTRIBUTE_SKIPDAYS ... 880
FEEDATTRIBUTE_SKIPHOURS ... 881
FEEDATTRIBUTE_TEXTINPUT ... 881
FEEDATTRIBUTE_TTL ... 881
FEEDATTRIBUTE_WEBMASTER ... 881
FEEDATTRIBUTE_XSL ... 882
FEEDAUTHTYPE_ALL ... 882

FEEDAUTHTYPE_ANONYMOUS 882
FEEDAUTHTYPE_DEFAULT 882
FEEDCACHETYPE_NONE 882
FEEDCACHETYPE_PRIVATE 883
FEEDCACHETYPE_PUBLIC 883
FEEDCACHETYPE_ROLE 883
FEEDFORMAT_ATOM10 883
FEEDSECUTYPE_PUBLIC 883
FEEDSECUTYPE_REALTIME 884
FEEDSECUTYPE_SELECTED 884
FEEDTYPE_DYNAMIC 884
FEEDTYPE_PREPUBLISHED 884
FIELDTYPE_CHARACTER 884
FIELDTYPE_DATE 885
FIELDTYPE_DATETIME 885
FIELDTYPE_LONGCHARACTER 885
FIELDTYPE_NUMBER 885
FIELDTYPE_SIGNEDNUMBER 885
FIELDTYPE_TIME 886
IBSOTYPE_ASYNC 886
IBSOTYPE_SYNC 886
IBSOTYPE_UNKNOWN 886
ICONURL_FEED_A 886
ICONURL_FEED_IA 887
INCREMENTALOPTION_NO 887
INCREMENTALOPTION_YES 887
LINKTYPE_FIRST 888
LINKTYPE_LAST 888
LINKTYPE_NEXT 888
LINKTYPE_PREVIOUS 889
MIMETYPE_ATOM 889
MIMETYPE_OPML 889
MIMETYPE_XML 890
OPERATINGMODE_AUTHORIZATION 890
OPERATINGMODE_DEFAULT 890
OPERATINGMODE_DELETION 890
OPERATINGMODE_EXECUTION 891
OPERATINGMODE_EXECUTION_NOENTRY 891
QUERYPARAMETER_CHILDFEEDID 891
QUERYPARAMETER_DATATYPEID 891
QUERYPARAMETER_DEFLOCALNODE 892
QUERYPARAMETER_DSSCOUNT 892
QUERYPARAMETER_DSSNAME 892
QUERYPARAMETER_DSSVALUE 892
QUERYPARAMETER_FEEDFORMAT 892

QUERYPARAMETER_FEEDID 893
QUERYPARAMETER_FEEDLIST 893
QUERYPARAMETER_FEEDTYPE 893
QUERYPARAMETER_IBTRANSID 893
QUERYPARAMETER_IFMODIFIEDSINCE 893
QUERYPARAMETER_IFNONEMATCH 894
QUERYPARAMETER_KEYWORD 894
QUERYPARAMETER_LANGUAGE 894
QUERYPARAMETER_NODENAME 894
QUERYPARAMETER_PAGENUM 894
QUERYPARAMETER_PORTALNAME 895
QUERYPARAMETER_PTPPB_SEARCH_MODE 895
QUERYPARAMETER_PTPPB_SEARCH_TEXT 895
RequestInfo 895
SF_MAXMINUTES_ALLMSGS 895
SF_MAXROWOPTION_ALLMSGS 896
SF_MAXROWOPTION_LATESTMSG 896
SF_PAGINGOPTION_NOPAGING 897
SF_PAGINGOPTION_SEGMENTED 897
SYSVAR_INVALID 898
USAGETYPE_ADMINSPECIFIED 898
USAGETYPE_FIXED 898
USAGETYPE_INTERNAL 898
USAGETYPE_NOTUSED 898
USAGETYPE_SYSVAR 899
USAGETYPE_USERSPECIFIED 899
XMLCHILDELEMENTS_CLOUD 899
XMLCHILDELEMENTS_PERSON 899
XMLCHILDELEMENTS_TEXTINPUT 900
XMLELEMENT_DAY 900
XMLELEMENT_DESCRIPTION 900
XMLELEMENT_DOMAIN 900
XMLELEMENT_EMAIL 901
XMLELEMENT_HOUR 901
XMLELEMENT_LINK 901
XMLELEMENT_NAME 901
XMLELEMENT_PATH 901
XMLELEMENT_PORT 902
XMLELEMENT_PROTOCOL 902
XMLELEMENT_REGISTERPROCEDURE 902
XMLELEMENT_TITLE 902
FeedDoc Class 902
FeedDoc Class Constructor 903
FeedDoc 903
FeedDoc Class Methods 903

- addCategory ... 903
- addEntry ... 904
- datetimeToString ... 905
- deleteCategory ... 906
- deleteEntry ... 907
- equals ... 907
- getEntry ... 908
- resetEntries ... 909
- stringToDatetime ... 909

FeedDoc Class Properties ... 910
- AllowMoreEntries ... 910
- ApplicationRelease ... 910
- Categories ... 910
- ContentUrl ... 911
- Copyright ... 911
- Description ... 911
- Entries ... 912
- Expires ... 912
- FeedFormat ... 912
- FeedUrl ... 913
- FirstUrl ... 913
- Generator ... 914
- ID ... 914
- LastUrl ... 914
- Logo ... 915
- MaxAge ... 915
- MaxEntries ... 915
- NextUrl ... 916
- ObjectType ... 916
- PreviousUrl ... 916
- RootElement ... 917
- Title ... 917
- Updated ... 917

FeedEntry Class ... 917

FeedEntry Class Constructor ... 918
- FeedEntry ... 918

FeedEntry Class Methods ... 918
- addCategory ... 919
- addContributor ... 919
- addEnclosure ... 920
- delete ... 921
- deleteCategory ... 922
- deleteContributor ... 922
- deleteEnclosure ... 923
- equals ... 924

FeedEntry Class Properties 924
Author 924
Categories 925
Comments 925
ContentUrl 925
Contributors 925
Copyright 926
Description 926
Enclosures 926
Expires 927
FeedDoc 927
FullContent 927
GUID 928
ID 928
MaxAge 915
ObjectType 929
Published 929
Title 929
Updated 929
Additional Feed Examples 930
Implementing a Real-Time Feed Request Handler 930

Chapter 18

Field Class 935

Understanding the Field Class 935
Considerations Using User Interface Properties 935
Shortcut Considerations 936
Data Type for a Field Object 936
Scope of a Field Object 936
Field Class Built-in Functions 936
Field Class Methods 937
AddDropDownItem 937
ClearDropDownList 938
DecryptPETKey 939
EncryptPETKey 940
GetAuxFlag 940
GetLongLabel 941
GetRelated 942
GetShortLabel 943
SearchClear 944
SetCursorPos 945
SetDefault 946

Field Class Properties 947
DataAreaCollapsed 947
DecimalPosition 948
DisplayFormat 948
DisplayOnly 950
DisplayZero 950
DisplayZeroChanged 950
EditError 951
Enabled 952
FieldLength 952
FormatLength 952
FormattedValue 953
HoverText 954
IsAltKey 954
IsAuditFieldAdd 954
IsAuditFieldChg 955
IsAuditFieldDel 955
IsAutoUpdate 955
IsChanged 955
IsDateRangeEdit 956
IsDescKey 956
IsDuplKey 956
IsEditTable 956
IsEditXlat 957
IsFromSearchField 957
IsInBuf 957
IsKey 958
IsListItem 958
IsNotUsed 958
IsRequired 959
IsRichTextEnabled 959
IsSearchItem 959
IsSystem 959
IsThroughSearchField 959
IsUseDefaultLabel 960
IsYesNo 960
Label 960
LabelImage 961
LongTranslateValue 962
MessageNumber 962
MessageSetNumber 963
MouseOverMsgNum 963
MouseOverMsgSet 964
Name 964
OriginalValue 965

ParentRecord 965
PromptTableName 966
SearchDefault 966
SearchEdit 966
SetComponentChanged 967
ShortTranslateValue 968
ShowRequiredFieldCue 969
SmartZero 969
SqlText 969
StoredFormat 970
Style 971
Type 971
Value 972
Visible 973

Chapter 19

File Class 975

Understanding File Layout 975
Data Type of a File Object 976
Scope of a File Object 976
File Security Considerations 977
File Access Interruption Recovery 977
Plain Text Files 978
Automatic PeopleCode Generation 980
End Of Line Considerations 980
File Layout Error Processing 981
Working With Relative Paths 982
File Class Built-in Functions 983
File Class Methods 983
Close 984
CreateRowset 984
Delete 986
GetPosition 986
GetString 988
Open 989
ReadLine 993
ReadRowset 994
SetFileId 996
SetFileLayout 998
SetPosition 1000
SetRecTerminator 1001
WriteLine 1002

WriteRaw 1003
WriteRecord 1004
WriteRowset 1006
WriteString 1008
File Class Properties 1009
CurrentRecord 1009
IgnoreInvalidId 1010
IsError 1010
IsNewFileId 1011
IsOpen 1012
Name 1012
TerminateLines 1012
UseSpaceForNull 1013
ZeroExtend 1016
File Layout Examples 1016
WriteRecord Example 1017
ReadRecord Example 1020
WriteRowset Example 1021
ReadRowset Example 1025
File Rowset Considerations 1026
Application Engine Example 1026
Multiple File Layouts 1029
Reading Multiple File Layouts 1030
Writing Multiple File Layouts 1032

Chapter 20

Grid Classes 1035

Understanding Grid and GridColumn Classes 1035
Shortcut Considerations 1036
The Grid Class in PeopleCode 1036
Data Type for a Grid or Grid Column Object 1037
Scope of a Grid or Grid Column Object 1037
Grid Class Built-in Function 1038
Grid Class Methods 1038
EnableColumns 1038
GetColumn 1039
LabelColumns 1041
SetProperties 1042
ShowColumns 1043
Grid Class Properties 1044
gridcolumn 1044
Label 1045

SummaryText 1045
GridColumn Class 1045
GridColumn Class Properties 1046
Enabled 1046
Label 1047
Name 1047
Visible 1048

Chapter 21

Internet Script Classes (iScript) 1051

Understanding Internet Script Classes 1051
URL vs. URI 1053
Web Libraries 1053
iScript Security 1054
When to Use an iScript 1055
Style Sheets and Styles 1056
Other Considerations 1056
Details of an iScript URL 1056
The Generate Functions 1057
Error Handling 1059
Scope of the Internet Script Classes 1059
Data Types of the iScript Classes 1060
Internet Script Classes 1060
Request Class 1060
Response 1061
Cookies 1061
Internet Script Classes Built-in Functions 1061
Request Class Methods 1062
GetContentBody 1062
GetCookieNames 1063
GetCookieValue 1064
GetHeader 1064
GetHeaderNames 1064
GetHelpURL 1065
GetParameter 1065
GetParameterNames 1065
GetParameterValues 1066
Request Class Properties 1066
AuthTokenDomain 1066
AuthType 1067
ByPassSignOn 1067
BrowserPlatform 1067

BrowserType 1067
BrowserVersion 1068
ContentURI 1068
ExpireMeta 1068
FullURI 1069
HTTPMethod 1069
LogoutURL 1070
PathInfo 1070
Protocol 1070
QueryString 1071
RelativeURL 1071
RemoteAddr 1071
RemoteHost 1071
RequestURI 1072
RemoteUser 1072
Scheme 1073
ServerName 1073
ServerPort 1073
ServletPath 1074
Timeout 1074
Response Class 1074
Response Class Methods 1074
Clear 1075
CreateCookie 1075
GetCookie 1075
GetCookieNames 1076
GetHeader 1076
GetHeaderNames 1076
GetImageURL 1077
GetJavaScriptURL 1077
GetStyleSheetURL 1078
RedirectURL 1079
SetContentType 1079
SetHeader 1080
UseSimpleURL 1080
Write 1081
WriteLine 1081
Response Class Properties 1082
Charset 1082
DefaultStyleSheetName 1082
Cookie Class 1083
Cookie Class Properties 1083
Domain 1083
MaxAge 1083
Name 1084

Path 1084
Secure 1084
Value 1084

Chapter 22

Java Class 1085

Understanding Java Class 1085
Supported Versions of Java 1086
Java Packages and Classes Delivered with PeopleTools 1086
System Setup for Java Classes 1086
From PeopleCode to Java 1088
State Management Concerns 1088
CreateJavaObject Example 1089
CreateJavaArray Example 1089
GetJavaClass Example 1090
From Java to PeopleCode 1090
SysVar Java Class 1091
SysCon Java Class 1091
Func Java Class 1091
Name Java Class 1091
Accessing PeopleCode Objects 1091
Using Application Classes From Java to PeopleCode 1094
PeopleCode and Java Data Types Mapping 1094
Considerations When Using the PeopleCode Java Functions 1096
Copying Arrays of Data Between PeopleCode and Java 1097
Considerations Working with the Java Garbage Collector 1097
Error Handling and the PeopleCode Java Functions 1098
The Java Debugging Environment 1098
Data Type of a Java Object 1099
Scope of a Java Object 1099
PeopleCode Java Built-in Functions 1099

Chapter 23

Mail Classes 1101

Understanding PeopleSoft MultiChannel Framework Mail Classes 1101
PeopleSoft MultiChannel Framework Mail Classes 1101
Scope of the Mail Classes 1102
Data Types of the Mail Classes 1102
Importing Mail Classes 1102
Creating a Mail Object 1103

Using the Mail Classes ... 1103
- General Lifecycle of an Outbound Email ... 1104
- Delivery Status Notification and Return Receipts ... 1105
- Priority ... 1105
- MCFBodyPart and MCFEmail Considerations ... 1105

Retrieving Email From a Mail Server With the MCFGetMail Class ... 1107
- Structure of Rowset Used for Email Information ... 1108
- Error Messages Returned by MCFGetMail Class Methods ... 1114

Mail Classes Constructors ... 1115
- MCFBodyPart ... 1115
- MCFGetMail ... 1116
- MCFOutboundEmail ... 1116
- MCFMailStore ... 1116
- SMTPSession ... 1116

MCFBodyPart Class ... 1116

MCFBodyPart Class Methods ... 1116
- AddHeader ... 1117
- GetHeader ... 1118
- GetHeaderCount ... 1118
- GetHeaderName ... 1119
- GetHeaderNames ... 1120
- GetHeaderValues ... 1120
- GetUnparsedHeaders ... 1121
- SetAttachmentContent ... 1122

MCFBodyPart Class Properties ... 1123
- AttachmentURL ... 1123
- Charset ... 1124
- ContentType ... 1124
- Description ... 1124
- Disposition ... 1125
- Filename ... 1125
- FilePath ... 1125
- FilePathType ... 1126
- IsAttachment ... 1126
- MultiPart ... 1126
- Text ... 1127

MCFEmail Class ... 1127

MCFEmail Class Properties ... 1127
- BCC ... 1128
- BounceTo ... 1128
- CC ... 1128
- From ... 1128
- Importance ... 1129
- Priority ... 1129
- Recipients ... 1130

RefIDs 1130
ReplyIDs 1130
ReplyTo 1130
Sender 1131
Sensitivity 1131
Subject 1131
Text 1131
MCFGetMail Class Methods 1132
CreateQuarantineFolder 1132
GetCount 1132
GetEmailCount 1133
ReadAllEmailHeadersWithAttach 1134
ReadEmails 1135
ReadEmailsWithAttach 1136
ReadEmailsWithUID 1137
ReadEmailWithAttach 1138
ReadHeaders 1139
RemoveEmail 1140
RemoveEmails 1141
SetMCFEmail 1142
MCFGetMail Class Properties 1142
AttachmentRoot 1143
ContentTypes 1143
ErrorCount 1143
IBNode 1144
MailServer 1144
Password 1145
QuarantineCount 1145
QuarantineFolder 1145
Status 1146
UserID 1146
MCFInboundEmail Class 1147
MCFInboundEmail Class Methods 1147
DumpToFile 1147
GetAttachments 1148
GetFrom 1149
GetParts 1149
GetSender 1150
ReadFromDatabase 1151
SaveToDatabase 1151
MCFInboundEmail Class Properties 1152
AttachList 1152
AttachSizes 1152
DttmReceived 1153
DttmSaved 1153

DttmSent 1153
IBNode 1154
Language 1154
MessageID 1154
NotifyCC 1154
NotifyTo 1155
OffsetReceived 1155
OffsetSent 1155
Server 1156
Size 1156
Status 1156
UID 1156
User 1156
MCFMailStore Class 1157
MCFMailStore Import Statements 1157
MCFMailStore Methods 1157
AuthorizeEmailAttach 1157
DeleteEmail 1158
RetrieveEmail 1159
StoreEmail 1160
MCFMailUtil Class 1161
MCFMailUtil Class Methods 1161
DecodeText 1161
DecodeWord 1163
EncodeText 1164
EncodeWord 1165
GetErrorMsgParam 1167
IsDomainNameValid 1168
IsEmailServerAvailable 1169
ParseRichTextHtml 1170
ValidateAddress 1171
MCFMailUtil Class Properties 1172
badaddresses 1173
ErrorDescription 1173
ErrorDetails 1173
ErrorMsgParamsCount 1174
imagesLocation 1174
MessageNumber 1174
MessageSetNumber 1175
MCFMultiPart Class 1175
MCFMultiPart Class Methods 1175
AddBodyPart 1175
GetBodyPart 1176
GetContentType 1177
GetCount 1177

MCFMultiPart Class Property 1177
SubType 1178
MCFOutboundEmail Class 1178
MCFOutboundEmail Class Methods 1178
AddAttachment 1178
AddHeader 1179
GetErrorMsgParam 1181
GetHeader 1181
GetHeaderCount 1182
GetHeaderName 1183
GetHeaderNames 1183
GetHeaderValues 1184
Send 1184
SetSMTPParam 1186
MCFOutboundEmail Class Properties 1186
BackupSMTPPort 1187
BackupSMTPServer 1187
BackupSMTPSSLClientCertAlias 1187
BackupSMTPSSLPort 1187
BackupSMTPUserName 1188
BackupSMTPUserPassword 1188
BackupSMTPUseSSL 1188
BCC 1189
BounceTo 1189
CC 1189
Charset 1189
ContentLanguage 1190
ContentType 1190
DefaultCharSet 1191
Description 1191
Disposition 1191
ErrorDescription 1192
ErrorDetails 1192
ErrorMsgParamsCount 1192
Filename 1193
FilePath 1193
FilePathType 1193
From 1194
Importance 1194
InvalidAddresses 1194
IsAuthenticationReqd 1195
IsOkToSendPartial 1195
IsReturnReceiptReqd 1195
MessageNumber 1196
MessageSetNumber 1196

MultiPart 1196
Priority 1197
Recipients 1197
RefIDs 1198
ReplyIDs 1198
ReplyTo 1198
ResultOfSend 1198
Sender 1199
Sensitivity 1199
SMTPPort 1200
SMTPServer 1200
SMTPSSLClientCertAlias 1200
SMTPSSLPort 1201
SMTPUserName 1201
SMTPUserPassword 1201
SMTPUseSSL 1202
StatusNotifyOptions 1202
StatusNotifyReturn 1203
Subject 1203
Text 1203
TimeToWaitForResult 1203
UsedDefaultConfig 1204
UsedPrimaryServer 1204
ValidSentAddresses 1204
ValidUnsentAddresses 1205
SMTPSession Class 1205
SMTPSession Class Methods 1205
Send 1205
SendAll 1206
SetSMTPParam 1207
SMTPSession Class Properties 1208
BackupPort 1208
BackupServer 1208
BackupSSLClientCertAlias 1209
BackupSSLPort 1209
BackupUserName 1209
BackupUserPassword 1210
BackupUseSSL 1210
IsAuthenticationReqd 1210
Port 1211
Server 1211
SSLClientCertAlias 1211
SSLPort 1212
UsedDefaultConfig 1212
UsedPrimaryServer 1212

UserName 1212
UserPassword 1213
UseSSL 1213
Mail Classes Examples 1214
Creating a Text Email 1214
Creating Email and Overriding SMTP Settings 1215
Creating an HTML Email 1216
Creating a Multipart Email with Text and HTML Parts 1217
Creating an HTML Email with Images 1219
Creating an Email from Rich Text Editor Output 1221
Creating an Email with Attachments 1227
Creating Email Attachments by Specifying URLs 1229
Creating and Sending Multiple Email Messages 1231
Sending an Email With Authentication 1231
Using an SSL Connection to Send Email 1232

Chapter 24

MCF IM Classes 1233

Understanding the MCFIMInfo Class 1233
Using the MCFIMInfo Class 1233
Data Type for MCFIMInfo Objects 1234
Scope of MCFIMInfo Objects 1234
Understanding the MCFIMSingleButton Class 1234
Importing the MCFIMSingleButton Class 1234
MCFIMInfo Class Built-in Functions 1234
MCFIMInfo Class Methods 1234
AddUser 1235
CheckAll 1235
GetAdditionalUserInfo 1236
GetErrorImageName 1236
GetOffLineImageName 1237
GetOnlineImageName 1237
GetUnknownImageName 1238
GetLaunchURL 1238
GetStatus 1239
RemoveUser 1239
MCFIMSingleButton Class Methods 1240
generateHTML 1240
generateJavaScript 1242
insertXMPPServerUserData 1242
MCFIMSingleButton 1243
updateXMPPServerUserData 1243

MCFIMSingleButton Class Properties 1244
hideDelay 1244

Chapter 25

Message Classes 1245

Understanding Message Classes 1245
Messages, Service Operations and Handlers 1246
Integration Broker Application Classes 1246
Data Types of Message Objects 1247
Scope of Message Objects 1247
Message Object Population 1248
Considerations When Populating a Rowset From a Message 1249
Considerations for Publishing and Subscribing to Partial Records 1251
Considerations When Subscribing to Character Fields 1251
Message Segments 1251
Content-Based Routing 1254
Error Handling 1256
Message Classes Reference 1257
Message Class Built-In Functions 1257
Message Class Methods 1257
Clone 1258
CopyPartRowset 1259
CopyRowset 1260
CopyRowsetDelta 1261
CopyRowsetDeltaOriginal 1263
CreateNextSegment 1266
DeleteSegment 1267
ExecuteEdits 1268
GenXMLPartString 1271
GenXMLString 1271
GetContentString 1272
GetDocument 1273
GetPartAliasName 1274
GetPartName 1274
GetPartRowset 1275
GetPartVersion 1276
GetPartXMLDoc 1276
GetQueryString 1277
GetRowset 1278
GetSegment 1280
GetXmlDoc 1280
InBoundPublish 1281

LoadXMLPartString 1282
LoadXMLString 1283
Publish 1283
SegmentRestart 1285
SetEditTable 1286
SetQueryString 1288
SetStatus 1289
SetXmlDoc 1290
SyncRequest 1291
Update 1293
UpdateSegment 1294
Message Class Properties 1294
ActionName 1295
AliasName 1295
ChannelName 1295
Cookies 1296
CurrentSegment 1297
DefaultMessageVersion 1297
DoNotPubToNodeName 1298
GUID 1298
IBException 1299
IBInfo 1299
IsActive 1299
IsBulkLoadTruncation 1300
IsDelta 1300
IsEditError 1301
IsEmpty 1302
IsLocal 1302
IsOperationActive 1303
IsParts 1303
IsPartsStructured 1303
IsRequest 1304
IsSourceNodeExternal 1304
IsStructure 1304
MessageDetail 1305
Name 1305
NRId 1305
OperationName 1305
OperationVersion 1306
ParentTransactionId 1306
PartCount 1306
PubID 1306
PubNodeName 1307
QueueName 1307
QueueSeqId 1308

- ResponseStatus 1308
- SegmentCount 1308
- SegmentsByDatabase 1308
- Size 1309
- SubName 1310
- SubQueueName 1310
- SubscriptionProcessId 1311
- TransactionId 1312
- Version 1312

IntBroker Class 1312

IntBroker Class Methods 1312

- Cancel 1312
- ConnectorRequest 1314
- ConnectorRequestUrl 1315
- DeleteOrphanedSegments 1316
- GetArchData 1316
- GetArchIBInfoData 1317
- GetIBInfoData 1318
- GetIBTransactionIDforAE 1319
- GetMessage 1319
- GetMessageErrors 1321
- GetMsgSchema 1321
- GetSyncIBInfoData 1322
- GetSyncLogData 1323
- InBoundPublish 1324
- IsOperationActive 1325
- MsgSchemaExists 1326
- Publish 1326
- Resubmit 1328
- SetMessageError 1329
- SetStatus 1331
- SwitchAsyncEventUserContext 1332
- SyncRequest 1332
- Update 1334
- UpdateXmlDoc 1335

IBInfo Class 1336

IBInfo Class Methods 1336

- AddAEAttribute 1336
- AddAttachment 1338
- AddAttribute 1339
- AddContainerAttribute 1340
- ClearAEAttributes 1341
- ClearAttachments 1342
- ClearAttributes 1342
- ClearContainerAttributes 1343

DeleteAEAttribute 1344
DeleteAttachment 1344
DeleteAttribute 1345
DeleteContainerAttribute 1346
GetAEAttributeName 1347
GetAEAttributeValue 1347
GetAttachmentContentID 1348
GetAttachmentProperty 1349
GetAttributeName 1351
GetAttributeValue 1352
GetContainerAttributeName 1352
GetContainerAttributeValue 1353
GetNumberofAEAttributes 1354
GetNumberOfAttributes 1354
GetNumberOfContainerAttributes 1355
GetTransactionIDforAE 1356
InsertAEResponseAttributes 1357
LoadConnectorProp 1357
LoadConnectorPropFromNode 1358
LoadConnectorPropFromRouting 1359
LoadConnectorPropFromTrx 1360
SetAttachmentProperty 1360
IBInfo Class Properties 1362
AppServerDomain 1362
CompressionOverride 1362
ConnectorOverride 1363
ConversationID 1363
DeliveryMode 1363
DestinationNode 1364
ExternalMessageID 1364
ExternalOperationName 1364
ExternalUserName 1364
ExternalUserPassword 1365
FinalDestinationNode 1365
FuturePublicationDateTime 1365
HTTPSessionId 1365
IBConnectorInfo 1366
InReplyToID 1366
MessageChannel 1366
MessageName 1366
MessageQueue 1367
MessageType 1367
MessageVersion 1367
NodeDN 1368
NonRepudiationID 1368

NumberOfAttachments 1368
OperationType 1368
OperationVersion 1369
OrigNode 1369
OrigProcess 1369
OrigTimeStamp 1369
OrigUser 1370
PublicationID 1370
RequestingNodeName 1370
RequestingNodeDescription 1370
ResponseAsAttachment 1370
SegmentsUnOrder 1371
SourceNode 1371
SyncServiceTimeout 1371
TransactionID 1372
UserName 1372
VisitedNodes 1372
WSA_Action 1373
WSA_FaultTo 1373
WSA_MessageID 1373
WSA_ReplyTo 1373
WSA_To 1374
IBConnectorInfo Collection 1374
IBConnectorInfo Collection Methods 1374
AddConnectorProperties 1374
AddQueryStringArg 1375
ClearConnectorProperties 1376
ClearQueryStringArgs 1377
DeleteConnectorProperties 1377
DeleteQueryStringArg 1378
GetConnectorPropertiesName 1378
GetConnectorPropertiesType 1379
GetConnectorPropertiesValue 1380
GetNumberOfConnectorProperties 1380
GetNumberOfQueryStringArgs 1381
GetQueryStringArgName 1381
GetQueryStringArgValue 1382
IBConnectorInfo Collection Properties 1382
ConnectorClassName 1382
ConnectorName 1383
Cookies 1383
PathInfo 1384
RemoteFrameworkURL 1384

Chapter 26

Mobile Classes **1385**

Mobile Classes Overview 1385
Understanding the Mobile Device and the Component Interface 1386
Mobile Classes Hierarchy 1390
Debugging in Mobile 1394
Debug Settings 1395
Trace Settings 1396
Error Handling 1396
Data Types for Mobile 1397
Scope of Mobile Objects 1397
Variables in Mobile PeopleCode 1397
Mobile Functions 1398
Session Class Method 1398
GetCompIntfc 1398
Mobile Class 1399
Mobile Class Methods 1399
Create 1399
Find 1400
Get 1401
GetName 1402
GetParent 1403
GetPeerDefaultAttributesByName 1404
GetPropertyAttrsByName 1404
GetPropertyByName 1405
GetPropertyInfoByName 1406
GetTopParent 1407
IsNew 1407
IsNewAndNeverSaved 1408
IsSameAs 1409
ReadBlobFromFile 1409
Save 1410
SetModified 1412
SetPropertyByName 1412
ValidateEnum 1413
ValuesDifferFromLastSync 1414
WasSaved 1415
WriteBlobToFile 1415
Mobile Class Property 1416
PropertyInfoCollection 1417
PropertyInfoCollection 1417

PropertyInfoCollection Method 1417
Item 1417
PropertyInfoCollection Property 1418
Count 1418
PropertyInfo Class 1419
PropertyInfo Class Properties 1419
DefaultValue 1419
HasDefaultValue 1420
IsCollection 1420
IsRefToParent 1421
IsPeerReference 1421
IsSimpleApplicationProperty 1421
IsSystemProperty 1422
Name 1422
PropertyInfoCollection 1423
CI Collection Class 1423
CI Collection Method 1424
Item 1424
CI Collection Properties 1424
Count 1425
DataCollection Class 1425
Data Collection Methods 1426
GetListViewAttrs 1426
Item 1427
DeleteItem 1428
InsertItem 1429
Data Collection Properties 1430
Count 1431
ListViewAttrs Class 1431
ListViewAttrs Method 1431
IsSet 1431
ListViewAttrs Properties 1432
DetailViewLabel 1432
Label 1433
ListViewLabel 1433
Visible 1434
PropertyAttrs Class 1434
PropertyAttrs Method 1435
IsSet 1435
PropertyAttrs Properties 1436
DisplayOnly 1436
InError 1437
Label 1437
ShowRequiredCue 1438
Visible 1438

PeerDefaultAttributes Class 1439
PeerDefaultAttributes Class Method 1439
IsSet 1439
PeerDefaultAttributes Properties 1440
DisplayOnly 1440
Label 1441
Visible 1441
Classes and Functions and System Variables Used in Mobile 1442
PeopleCode Classes Used in Mobile 1442
System Variables Used in Mobile 1442
Built-In Functions Used in Mobile 1443

Chapter 27

Notification Classes 1445

Understanding Notification Classes 1445
Scope of the Notification Classes 1445
Data Types of the Notification Classes 1446
How to Import the Notification Classes 1446
How to Create a Notification Object 1447
Notification Classes Constructors 1447
Notification 1447
NotificationAddress 1448
NotificationTemplate 1450
Worklist 1451
WorklistEntry 1451
WSWorklistEntry 1452
Notification Class 1453
Notification Class Import Statements 1453
Notification Class Method 1453
Send 1453
Notification Class Properties 1454
ContentType 1454
dttmCreated 1454
EmailReplyTo 1455
FileNames 1455
FileTitles 1455
language_cd 1456
Message 1456
NotifyBCC 1456
NotifyCC 1456
NotifyFrom 1457
NotifyGuid 1457

NotifyTo 1457
Rte 1458
SourceComponent 1458
SourceMarket 1458
SourceMenu 1458
Subject 1458
Template 1459
NotificationAddress Class 1459
NotificationAddress Class Properties 1459
Channel 1459
Description 1460
EmailId 1460
Language 1460
Oprid 1460
NotificationTemplate Class 1460
NotificationTemplate Class Methods 1461
GetAndExpandTemplate 1461
SetupCompVarsAndRcpts 1462
SetupGenericVars 1463
NotificationTemplate Class Properties 1464
ComponentId 1464
Instruction 1464
Language 1465
Market 1465
Priority 1465
Responses 1465
Subject 1465
TemplateId 1466
TemplateType 1466
Text 1466
Worklist Class 1466
Worklist Class Method 1467
Reassign 1467
WorklistEntry Class 1467
WorklistEntry Class Methods 1468
Create 1468
GetResponseStatus 1469
Reassign 1470
Save 1471
SaveWithCustomData 1472
SelectByKey 1473
SelectByMessageId 1474
Update 1475
WorklistEntry Class Properties 1476
actiondttm 1476

busactivity 1476
buseventname 1476
busprocname 1476
commentshort 1477
do_replicate_flag 1477
instanceid 1477
instselecteddttm 1477
inststatus 1478
insttimeoutdttm 1478
instworkeddttm 1478
IsCreatedViaWebService 1478
oprid 1479
originatorid 1479
prevoprid 1479
requestmessageid 1479
ResponseStatus 1479
syncid 1480
timedout 1480
transactionid 1480
url 1481
wl_priority 1481
wldaystoselect 1481
wldaystowork 1481
worklistname 1482
WSWorklistEntry Class 1482
WSWorklistEntry Class Methods 1482
OnError 1482
OnNotify 1483
WSWorklistEntry Class Properties 1483
InstanceID 1484
TransactionID 1484
Notification Classes Examples 1484
Creating a WorklistEntry 1484
Updating a WorklistEntry 1486

Chapter 28

Optimization PeopleCode 1489

Using Optimization PeopleCode on the Application Server 1489
Using Optimization PeopleCode in an Application Engine Program 1490
Performing Optimization in PeopleCode 1490
Creating New Analytic Instances 1491
Loading Analytic Instances Into an Analytic Server 1491

Running Optimization Transactions 1492
Invoking the Optimization PeopleCode Plug-In 1493
Shutting Down Optimization Engines 1494
Deleting Existing Analytic Instances 1494
Programming for Database Updates 1495
Using Lights-Out Mode with Optimization 1495
Understanding Lights-out Mode 1495
Creating a Request Message 1497
Creating a Response Message 1501
Editing the Request PeopleCode 1502
Editing the Response PeopleCode 1506
Optimization Built-in Functions 1508
CreateOptEngine 1508
CreateOptInterface 1510
DeleteOptProbInst 1511
GetOptEngine 1513
GetOptProbInstList 1514
InsertOptProbInst 1515
IsValidOptProbInst 1517
OptEngine Class Methods 1518
CheckOptEngineStatus 1518
FillRowset 1520
GetDate 1522
GetDateArray 1523
GetDateTime 1524
GetDateTimeArray 1525
GetNumber 1526
GetNumberArray 1527
GetString 1528
GetStringArray 1529
GetTime 1531
GetTimeArray 1532
GetTraceLevel 1533
RunAsynch 1534
RunSynch 1536
SetTraceLevel 1537
ShutDown 1539
OptEngine Class Properties 1540
DetailMsgs 1541
DetailedStatus 1542
OptBase Application Class 1543
OptBase Class Methods 1544
GetParmDate 1544
GetParmDateArray 1545
GetParmDateTime 1546

GetParmDateTimeArray 1546
GetParmNumber 1547
GetParmNumberArray 1548
GetParmInt 1548
GetParmIntArray 1549
GetParmString 1550
GetParmStringArray 1550
GetParmTime 1551
GetParmTimeArray 1552
Init 1552
OptDeleteCallback 1553
OptInsertCallback 1554
OptPostUpdateCallback 1554
OptPreUpdateCallback 1555
OptRefreshCallback 1556
SetOutputParmDate 1557
SetOutputParmDateArray 1557
SetOutputParmDateTime 1558
SetOutputParmDateTimeArray 1559
SetOutputParmNumber 1559
SetOutputParmNumberArray 1560
SetOutputParmInt 1561
SetOutputParmIntArray 1561
SetOutputParmString 1562
SetOutputParmStringArray 1563
SetOutputParmTime 1563
SetOutputParmTimeArray 1564
OptInterface Class Methods 1565
ActivateModel 1565
ActivateObjective 1566
DeactivateModel 1567
DumpMsgToLog 1567
FindRowNum 1568
GetSolution 1569
GetSolutionDetail 1571
IsModelActive 1573
RestoreBounds 1573
SetVariableBounds 1574
SetVariableType 1576
Solve 1577

Chapter 29

Page Class **1581**

Understanding Page Class 1581
Shortcut Considerations 1581
Data Type of a Page Object 1582
Scope of a Page Object 1582
Page Class Built-in Function 1582
Page Class Properties 1582
- CopyURLLink 1582
- CustomizePageLink 1583
- DisplayOnly 1583
- HelpLink 1584
- Name 1584
- NewWindowLink 1585
- Visible 1585

Chapter 30

Portal Registry Classes **1587**

Understanding the Portal Registry 1587
- Folders 1588
- Content References 1588
- Nodes 1589
- Security 1589
- Attributes 1590
- Additional Portal Objects 1590

Using the PortalRegistry API 1591
Using Security 1591
- Using Object Properties 1591
- Accessing Objects 1592
- Using PermissionValue, RolePermissionValue, Cascading Permissions and CascadingRolePermissions 1592

Working With ValidFrom and ValidTo 1595
Doing Error Handling 1595
Understanding the Life Cycle of a PortalRegistry Object 1596
Viewing the PortalRegistry Class Hierarchy 1598
Using Content References 1604
- UsageType 1604
- TemplateType 1606
- URLType 1606

Nodes and URL ... 1607
Naming Conventions ... 1608
Deleting Content Considerations ... 1609
Saving Content Considerations ... 1609
Data Type of a PortalRegistry Object ... 1610
Scope of a PortalRegistry Object ... 1610
Portal Registry Reference ... 1611
Session Class Methods ... 1611
FindPortalRegistries ... 1611
GetActualRemoteNodes ... 1612
GetLocalNode ... 1613
GetNodes ... 1613
GetPortalRegistry ... 1614
GetRemoteNodes ... 1615
PortalRegistry Class ... 1615
PortalRegistry Class Methods ... 1616
Close ... 1616
CopyObject ... 1617
Create ... 1618
CreateContentRefLink ... 1619
CreateRemote ... 1620
Delete ... 1621
DeleteHomepage ... 1622
FindCRefByName ... 1623
FindCRefByURL ... 1624
FindCRefForURL ... 1625
FindCRefLinkByName ... 1627
FindFolderByName ... 1627
FindPgltByName ... 1629
GetAbsoluteContentURL ... 1630
GetDefaultHPTabOID ... 1630
GetQualifiedURL ... 1631
GrantPermissionForComponent ... 1631
GrantPermissionForScript ... 1633
Open ... 1634
PermissionListDelete ... 1635
PermissionListSaveAs ... 1635
RevokePermissionForComponent ... 1636
RevokePermissionForScript ... 1637
Save ... 1638
PortalRegistry Class Properties ... 1639
DefaultTemplate ... 1639
Description ... 1640
Favorites ... 1640
FolderNavObject ... 1640

Homepage 1640
IsFolderNavigation 1641
Name 1641
NodeTemplates 1641
OwnerId 1641
PageletCategories 1642
Portals 1642
RootFolder 1642
TabDefinitions 1642
TemplateObject 1643
PortalRegistry Collection 1643
PortalRegistry Collection Methods 1643
First 1643
Item 1644
Next 1644
PortalRegistry Collection Property 1645
Count 1645
Node Class 1645
Node Class Properties 1646
ActiveNode 1646
AppsRelease 1646
ContentURI 1646
DefaultPortalName 1646
Description 1647
IsDefault 1647
IsLocal 1647
Name 1648
NodePassword 1648
NodeType 1648
PortalURI 1649
ToolsRelease 1649
Node Collection 1649
Node Collection Methods 1649
First 1649
ItemByName 1650
Next 1651
Node Collection Property 1651
Count 1651
RemoteNode Collection 1652
RemoteNode Collection Methods 1652
First 1652
ItemByName 1652
Next 1653
RemoteNode Collection Property 1654
Count 1654

Portal Class 1654
Portal Class Method 1654
    Save 1654
Portal Class Properties 1655
    HostNodeName 1655
    IsLocal 1655
    Name 1655
Portal Collection 1656
Portal Collection Methods 1656
    First 1656
    ItemByName 1656
    Next 1657
Portal Collection Properties 1658
    Count 1658
NodeTemplate Class 1658
NodeTemplate Class Properties 1658
    DefaultTemplate 1658
    Name 1659
    TemplateObject 1659
NodeTemplate Collection 1659
NodeTemplate Collection Methods 1659
    DeleteItem 1659
    First 1660
    InsertItem 1661
    ItemByName 1661
    Next 1662
NodeTemplate Collection Properties 1662
    Count 1663
Folder Class 1663
Folder Class Method 1663
    Save 1663
Folder Class Properties 1664
    AbnContentProvider 1664
    AbnDataSource 1665
    AbnPeopleCode 1665
    Attributes 1666
    Author 1666
    AuthorAccess 1666
    Authorized 1667
    CascadedPermissions 1667
    CascadedRolePermissions 1668
    ContentRefs 1668
    CreationDate 1668
    DefaultChartNavigation 1669
    Description 1669

Folders 1669
IsMobile 1670
IsVisible 1670
Label 1670
Name 1671
OwnerId 1671
ParentName 1671
Path 1671
Permissions 1672
Product 1672
PublicAccess 1672
RolePermissions 1673
SequenceNumber 1673
TreeEffectiveDate 1673
TreeName 1674
TreeSetId 1675
TreeStructureName 1676
TreeUserKeyValue 1677
ValidFrom 1678
ValidTo 1678
Folder Collection 1678
Folder Collection Methods 1678
DeleteItem 1679
First 1679
InsertItem 1680
ItemByName 1681
Next 1681
Folder Collection Property 1682
Count 1682
Content Reference Class 1683
Content Reference Class Methods 1683
CreateLink 1683
Save 1684
Content Reference Class Properties 1685
AbsoluteContentURL 1685
AbsolutePortalURL 1685
AssignedPagelets 1686
Attributes 1686
Author 1686
AuthorAccess 1687
Authorized 1687
CascadedPermissions 1687
CascadedRolePermissions 1688
ContentProvider 1688
CreationDate 1689

Data 1689
Description 1689
HtmlText 1690
IsMobile 1690
IsVisible 1690
Label 1691
Links 1691
Name 1691
OwnerId 1691
ParentName 1692
Path 1692
Permissions 1692
Product 1693
PublicAccess 1693
QualifiedURL 1693
RelativeURL 1693
RolePermissions 1694
SequenceNumber 1694
StorageType 1695
Template 1696
TemplateObject 1696
TemplateType 1697
URL 1697
URLType 1698
UsageType 1699
ValidFrom 1699
ValidTo 1700
Content Reference Collection 1700
Content Reference Collection Methods 1700
DeleteItem 1700
First 1701
InsertItem 1702
ItemByName 1703
Next 1703
Content Reference Collection Property 1704
Count 1704
AttributeValue Class 1704
AttributeValue Class Properties 1705
Label 1705
Name 1705
Translatable 1706
Value 1706
Attribute Collection 1707
Attribute Collection Methods 1707
DeleteItem 1707

First 1708
InsertItem 1708
ItemByName 1709
Next 1710
Attribute Collection Property 1710
Count 1711
PermissionValue Class 1711
PermissionValue Class Properties 1711
Cascade 1711
Name 1712
PermType 1712
PermissionValue Collection 1712
PermissionValue Collection Methods 1713
DeleteItem 1713
First 1714
InsertItem 1714
ItemByName 1715
Next 1716
PermissionValue Collection Property 1716
Count 1716
RolePermissionValue Collection 1717
RolePermissionValue Collection Methods 1717
DeleteItem 1717
First 1718
InsertItem 1719
ItemByName 1719
Next 1720
RolePermissionValue Collection Property 1721
Count 1721
ContentReference Links 1721
ContentReference Links Methods 1722
Delete 1722
Save 1722
ContentReference Links Properties 1723
AbsoluteContentURL 1723
AbsolutePortalURL 1724
Attributes 1724
Author 1724
AuthorAccess 1724
Authorized 1725
CascadedPermissions 1725
CascadedRolePermissions 1726
ContentProvider 1726
CreationDate 1726
Data 1726

Description 1727
IsMobile 1727
IsVisible 1727
Label 1728
Name 1728
OwnerId 1728
ParentName 1729
Path 1729
Permissions 1729
Product 1730
PublicAccess 1730
RelativeURL 1730
RolePermissions 1731
SequenceNumber 1731
Template 1731
TemplateObject 1732
TemplateType 1732
URL 1733
URLType 1733
UsageType 1733
ValidFrom 1734
ValidTo 1734
Link Collection 1734
Link Collection Methods 1734
First 1735
Next 1735
Link Collection Property 1735
Count 1736
Link Class 1736
Link Properties 1736
LinksObjectName 1736
LinksObjectType 1736
LinksPortalName 1737
TabDefinition Class 1737
TabDefinition Class Methods 1737
Save 1737
TabDefinition Class Properties 1738
AssignedPagelets 1738
AvailableCategories 1738
AvailablePagelets 1739
Attributes 1739
Author 1739
AuthorAccess 1739
ColumnLayout 1740
CreationDate 1740

Description 1740
DynamicCategories 1740
HelpID 1741
HtmlText 1741
IsHideActionBar 1741
IsLayoutLocked 1741
IsRenameable 1742
Label 1742
LayoutBehavior 1742
Name 1743
OwnerId 1743
Product 1743
PublicAccess 1743
QualifiedURL 1744
SequenceNumber 1744
StyleSheet 1744
ValidFrom 1744
ValidTo 1745
TabDefinition Collection 1745
TabDefinition Collection Methods 1745
DeleteItem 1745
First 1746
InsertItem 1747
ItemByName 1747
Next 1748
TabDefinition Collection Property 1748
Count 1749
AssignedPagelet Class 1749
AssignedPagelet Class Properties 1749
Column 1749
LayoutBehavior 1749
PageletName 1750
Row 1750
AssignedPagelet Collection 1750
AssignedPagelet Collection Methods 1751
DeleteItem 1751
First 1751
InsertItem 1752
ItemByName 1753
Next 1754
AssignedPagelet Collection Property 1754
Count 1754
AvailableCategory Class 1755
AvailableCategory Class Properties 1755
AvailablePagelets 1755

CategoryName 1755
AvailableCategory Collection 1756
AvailableCategory Collection Methods 1756
First 1756
ItemByName 1757
Next 1757
AvailableCategory Collection Property 1758
Count 1758
AvailablePagelet Class 1758
AvailablePagelet Class Properties 1758
CategoryLabel 1759
CategoryName 1759
Column 1759
LayoutBehavior 1759
PageletLabel 1760
PageletName 1760
Row 1760
AvailablePagelet Collection 1760
AvailablePagelet Collection Methods 1761
First 1761
ItemByName 1761
Next 1762
AvailablePagelet Collection Property 1762
Count 1763
DynamicCategory Collection 1763
DynamicCategory Collection Methods 1763
DeleteItem 1763
First 1764
InsertItem 1765
ItemByName 1765
Next 1766
DynamicCategory Collection Property 1767
Count 1767
PageletCategory Class 1767
PageletCategory Class Method 1767
Save 1767
PageletCategory Class Properties 1768
Attributes 1768
Author 1769
AuthorAccess 1769
Authorized 1769
CascadedPermissions 1769
CreationDate 1770
Description 1770
Label 1770

- Name 1771
- OwnerId 1771
- Pagelets 1771
- Permissions 1772
- Product 1772
- PublicAccess 1772
- SequenceNumber 1773

PageletCategory Collection 1773

PageletCategory Collection Methods 1773
- DeleteItem 1773
- First 1774
- InsertItem 1775
- ItemByName 1775
- Next 1776

PageletCategory Collection Property 1776
- Count 1777

Pagelet Class 1777

Pagelet Class Method 1777
- Save 1777

Pagelet Class Properties 1778
- Attributes 1778
- Author 1778
- AuthorAccess 1779
- Authorized 1779
- CascadedPermissions 1779
- ContentProvider 1779
- CreationDate 1780
- DefaultColumn 1780
- Description 1780
- EditPageContentProvider 1780
- EditPageQueryString 1781
- HelpID 1781
- IsHideMinimize 1781
- Label 1781
- Name 1782
- OwnerId 1782
- ParentName 1782
- Permissions 1782
- Product 1783
- PublicAccess 1783
- QualifiedURL 1783
- SequenceNumber 1783
- URL 1784
- URLType 1784

Pagelet Collection 1785

Pagelet Collection Methods 1785
DeleteItem 1785
First 1786
InsertItem 1786
ItemByName 1787
Next 1788
Pagelet Collection Property 1788
Count 1788
UserHomepage Class 1789
UserHomepage Class Method 1789
Save 1789
UserHomepage Class Properties 1790
Greeting 1790
UserId 1790
UserTab 1790
UserTab Class 1791
UserTab Class Properties 1791
ColumnLayout 1791
Label 1791
QualifiedURL 1792
SelectedPagelets 1792
SequenceNumber 1792
TabName 1792
UserTab Collection 1793
UserTab Collection Methods 1793
DeleteItem 1793
First 1794
InsertItem 1794
ItemByName 1795
Next 1795
UserTab Collection Property 1796
Count 1796
SelectedPagelet Class 1796
SelectedPagelet Class Properties 1797
CategoryName 1797
Column 1797
IsMinimized 1797
PageletName 1798
Row 1798
SelectedPagelet Collection 1798
SelectedPagelet Collection Methods 1798
DeleteItem 1798
First 1799
InsertItem 1800
ItemByName 1800

Next 1801
SelectedPagelet Collection Property 1801
Count 1802
Favorite Class 1802
Favorite Class Properties 1802
CRefName 1802
Label 1803
QualifiedURL 1803
SequenceNumber 1803
URL 1803
Favorite Collection 1804
Favorite Collection Methods 1804
DeleteItem 1804
First 1805
InsertItem 1805
ItemByLabel 1806
Next 1807
Save 1807
Favorite Collection Property 1808
Count 1808
PortalRegistry Classes Example 1808
Changing PortalRegistry Properties 1808
Adding a ContentProvider 1810
Adding a Folder 1811
Adding a Content Reference 1814
Setting Permissions Using the PermissionValue Object 1817
Using Attributes 1819
Visual Basic Example 1820
C/C++ Example 1820

Chapter 31

PostReport Class 1825

PostReport Class Overview 1825
Determining the Distribution List 1825
Data Type of a PostReport Object 1826
Scope of a PostReport Object 1826
PostReport Class Built-in Function 1826
PostReport Class Methods 1826
AddDistributionOption 1827
Put 1827
PostReport Class Properties 1828
ExpirationDate 1828

OutDestFormat 1829
ProcessInstance 1829
ProcessName 1830
ProcessType 1830
ReportDescr 1830
ReportFolder 1831
ReportId 1831
ServerName 1832
SourceReportPath 1832
PostReport Class Examples 1832

Chapter 32

Process Request Classes 1835

Understanding Process Request Classes 1835
Data Type of a ProcessRequest Object 1836
Scope of a ProcessRequest Object 1836
Options for Items In Jobs and Jobsets 1836
Values for Output Type and Format 1841
Alternative Options to Specify Email or Web Attributes 1844
File Dependant Processing 1845
ProcessRequest Class Built-in Functions 1845
ProcessRequest Class Methods 1846
AddDistributionOption 1846
AddNotifyInfo 1847
PrintJobHTMLRpt 1848
PrintJobRqstRpt 1849
PrintSchdlHTMLRpt 1852
RunJobSetNow 1855
Schedule 1855
SetEmailOption 1857
SetItemFolder 1858
SetNotifyAppMethod 1859
SetNotifyService 1862
SetOutputOption 1863
UpdateRunStatus 1865
ProcessRequest Class Properties 1865
EmailAttachLog 1865
EmailSubject 1866
EmailText 1866
EmailWebReport 1867
FileName 1867
JobName 1868

LanguageCd ... 1868
NotifyTextMsgNum ... 1868
NotifyTextMsgSet ... 1869
OutDest ... 1869
OutDestFormat ... 1871
OutDestType ... 1871
PortalFolder ... 1872
ProcessInstance ... 1872
ProcessName ... 1872
ProcessType ... 1873
RunControlID ... 1874
RunDateTime ... 1874
RunLocation ... 1875
RunRecurrence ... 1875
RunStatus ... 1875
Status ... 1877
TimeZone ... 1877
ProcessRequest Class Examples ... 1877
Scheduling a Single Process ... 1878
Scheduling a Job Where Job Item Changes Are Not Made ... 1878
Scheduling a Job Where Job Item Changes Are Made ... 1879
PrcsApi Class ... 1880
Scope of a PrcsApi Object ... 1881
Data Type of a PrcsApi Object ... 1881
How to Import the PrcsApi Class ... 1881
How to Create a PrcsApi Object ... 1881
PrcsApi Class Constructor ... 1882
PrcsApi ... 1882
PrcsApi Class Methods ... 1882
getAllFileNames ... 1882
notifyToWindow ... 1883

Chapter 33

Query Classes ... 1885

Understanding Query Classes ... 1885
Collections in the Query Classes ... 1886
Life Cycle of a Query ... 1887
Query Classes Hierarchy ... 1888
Query API Overview ... 1891
Query ... 1892
QuerySelect ... 1892
QueryRecords ... 1893

- QueryOutputFields 1893
- QuerySelectedFields 1893
- QueryCriteria 1893
- QueryDBRecords and QueryDBRecordFields 1894

Working With Query Criteria and Expressions 1894

- Setting the Expression Type 1894
- Adding New Expressions 1895
- Adding an Operator and Expression 2 Dependencies 1895
- Setting a Drilling URL 1896

Query Monitor 1898

Using Query Metadata 1898

Running a Query 1900

- Specifying the User's Language 1901

Query Security 1901

Error Handling With Query Classes 1901

Understanding QueryOutputFields and QuerySelectedFields 1902

Understanding Having Criteria 1903

Data Type of Query Objects 1903

Scope of Query Objects 1903

Query Classes Reference 1904

Session Class Methods in the Query API 1905

- AdvancedSearchQueries 1905
- AdvancedSearchRecords 1908
- FindQueryDBRecords 1910
- FindQueries 1910
- FindQueriesDateRange 1911
- GetQuery 1912
- GetQuerySecurityProfile 1913
- SearchQueryDBRecords 1914
- SearchPrivateQueries 1915
- SearchPublicQueries 1917

Query Collection 1918

Query Collection Methods 1919

- First 1919
- Item 1920
- ItemByName 1920
- Next 1921

Query Collection Property 1921

- Count 1922

Query Class 1922

Query Class Methods 1922

- AddPrompt 1922
- AddQuerySelect 1923
- AddTrackingURL 1924
- Close 1925

CopyPrivateQuery 1925
Create 1926
Delete 1928
DeletePrompt 1929
FindExpression 1929
FormatBinaryResultString 1930
FormatResultString 1931
GetTreePromptCount 1932
Open 1933
Rename 1934
RunToFile 1935
RunToRowset 1938
RunToString 1941
Save 1944
SetTrackingURL 1944
Query Class Properties 1945
Approved 1946
ApproveOprId 1946
ApproveUserId 1946
ApproveDtTm 1947
CreateOprId 1947
CreateDtTm 1947
CreateUserId 1947
Description 1948
Disabled 1948
ExecAppName 1948
ExecLogging 1949
Folder 1949
LastSQLErrorCode 1949
LastUpdDttm 1949
LastUpdOprId 1950
LongDescription 1950
Metadata 1950
MetaSQL 1950
MoreRowsAvailable 1951
Name 1951
OutputUnicode 1951
PDFFont 1952
Prompts 1953
PromptRecord 1953
PublicPrivate 1953
QuerySelect 1954
QueryStatistics 1954
RunTimePrompts 1954
SQL 1955

Type 1955
QuerySelect Collection 1956
QuerySelect Collection Methods 1956
AddUnion 1956
DeleteQuerySelect 1957
First 1957
Item 1958
ItemBySelNum 1958
Next 1959
QuerySelect Collection Property 1960
Count 1960
QuerySelect Class 1960
QuerySelect Methods 1961
AddAllFields 1961
AddCriteria 1962
AddExpression 1963
AddHavingCriteria 1963
AddQueryOutputField 1964
AddQueryRecord 1965
AddQuerySelectedField 1965
DeleteCriteria 1966
DeleteExpression 1967
DeleteField 1968
DeleteHavingCriteria 1968
DeleteRecord 1969
QuerySelect Properties 1970
Criteria 1970
Distinct 1970
Expressions 1970
HavingCriteria 1971
ParentSelectNum 1971
QueryOutputFields 1971
QueryRecords 1972
QuerySelectedFields 1972
QuerySelects 1972
SelectNum 1973
SelectType 1973
QueryRecord Collection 1973
QueryRecord Collection Methods 1973
First 1974
Item 1974
ItemByAlias 1975
Next 1975
QueryRecord Collection Property 1976
Count 1976

QueryRecord Class 1976
QueryRecord Class Method 1977
GetField 1977
QueryRecord Class Properties 1977
Description 1978
JoinAlias 1978
JoinFieldName 1978
JoinType 1978
Name 1979
QueryFields 1979
RecordAlias 1979
QueryField Collection 1980
QueryField Collection Methods 1980
First 1980
Item 1981
ItemByNameAndAlias 1981
ItemByExpNum 1982
Next 1983
QueryField Collection Property 1983
Count 1983
QueryField Class 1984
QueryField Class Methods 1984
AddTranslateExpression 1984
AddTranslateField 1985
GetImageFormat 1985
QueryField Class Properties 1986
Aggregate 1986
ColumnNumber 1987
Description 1987
Decimal 1987
ExpNum 1987
Flag 1988
Format 1988
HeadingText 1989
HeadingType 1989
HeadingUniqueFieldName 1989
Length 1990
LongName 1990
Name 1990
OrderByDirection 1990
OrderByNumber 1991
QueryRecord 1991
RecordAlias 1992
ShortName 1992
TranslateEffDtLogic 1992

- TranslateExpression 1993
- TranslateField 1993
- TranslateOption 1993
- Type 1994

QueryCriteria Collection 1994

QueryCriteria Collection Methods 1995
- First 1995
- Item 1995
- ItemByName 1996
- Next 1997

QueryCriteria Collection Property 1997
- Count 1997

QueryCriteria Class 1998

QueryCriteria Class Methods 1998
- AddExpr1Expression 1998
- AddExpr1Field 1999
- AddExpr2Expression 2000
- AddExpr2Field1 2000
- AddExpr2Field2 2001
- AddExpr2List 2002
- AddExpr2Subquery 2002

QueryCriteria Class Properties 2003
- Expr1Expression 2003
- Expr1Field 2003
- Expr1Type 2004
- Expr2Constant1 2005
- Expr2Constant2 2005
- Expr2Expression1 2005
- Expr2Expression2 2006
- Expr2Field1 2006
- Expr2Field2 2006
- Expr2List 2007
- Expr2Subquery 2007
- Expr2Type 2007
- Logical 2011
- LParenLvl 2011
- Name 2011
- Negation 2012
- Operator 2012
- R1ExprNum 2014
- R2ExprNum 2014
- R1ExprType 2014
- R2ExprType 2015
- RParenLvl 2015

QueryExpression Collection 2015

QueryExpression Collection Methods 2015
First 2015
Item 2016
ItemByName 2017
Next 2017
QueryExpression Collection Property 2018
Count 2018
QueryExpression Class 2018
QueryExpression Class Properties 2019
Aggregate 2019
BindFlag 2019
Decimal 2020
ExpNum 2020
IsXlatExpression 2020
Length 2020
Name 2020
OutputField 2021
RightExprFlag 2021
SelectedField 2021
Text 2021
Type 2022
QueryList Class 2023
QueryList Class Methods 2023
AddListValue 2023
First 2024
Item 2024
Next 2025
QueryList Class Property 2025
Count 2026
QueryListValue Class 2026
QueryListValue Class Properties 2026
IsPrompt 2026
Value 2026
QueryRecordHierarchy Collection 2027
QueryRecordHierarchy Collection Methods 2027
First 2027
Item 2028
ItemByName 2028
Next 2029
QueryRecordHierarchy Collection Property 2030
Count 2030
QueryRecordHierarchy Class 2030
QueryRecordHierarchy Class Properties 2030
Description 2030
Level 2031

Name 2031
ParentFlag 2031
Query Metadata Collection 2031
Query Metadata Collection Methods 2032
First 2032
Item 2032
ItemByName 2033
Next 2034
Query Metadata Collection Property 2034
Count 2034
Query Metadata Class 2035
Query Metadata Class Properties 2035
Name 2035
Value 2036
QueryStatistics Class 2036
QueryStatistics Class Properties 2036
AvgExecTime 2036
AvgFetchTime 2036
AvgNumRows 2037
ExecCount 2037
LastExecDtTm 2037
QuerySecurityProfile Class 2037
QuerySecurityProfile Class Properties 2037
AllowAnyJoin 2038
AllowDistinct 2038
AllowExpressions 2038
AllowSubqueries 2038
AllowUnions 2038
ApprovePrivateQuery 2039
ApprovePublicQuery 2039
CanCreatePublic 2039
CanCreateWorkFlow 2039
CanModifyQuery 2040
CanRunQuery 2040
CanRunToCrystal 2040
CanRunToExcel 2040
LimitUnapproved 2041
MaxInTreeCriteria 2041
MaxJoins 2041
MaxRowsToFetch 2041
MaxUnapprovedRows 2042
QueryDBRecord Collection 2042
QueryDBRecord Collection Methods 2042
First 2042
Item 2043

ItemByName 2043
Next 2044
QueryDBRecord Collection Property 2045
Count 2045
QueryDBRecord Class 2045
QueryDBRecord Class Methods 2045
QueryDBRecordFieldByIndex 2045
QueryDBRecordFieldByName 2046
QueryDBRecord Class Properties 2047
Description 2047
Name 2047
QueryDBRecordFields 2047
RecordHierachy 2047
QueryDBRecordField Collection 2048
QueryDBRecordField Collection Methods 2048
First 2048
Item 2049
ItemByName 2049
Next 2050
Sort 2051
QueryDBRecordField Collection Property 2051
Count 2052
QueryDBRecordField Class 2052
QueryDBRecordField Class Method 2052
GetImageFormat 2052
QueryDBRecordField Class Properties 2053
Decimal 2053
Description 2053
Flag 2054
Format 2055
Length 2056
LongName 2056
LookupTableName 2056
LookupTableRecord 2056
Name 2056
Shortname 2057
Type 2057
QueryPrompt Collection 2058
QueryPrompt Collection Methods 2058
First 2058
Item 2059
ItemByName 2059
Next 2060
QueryPrompt Collection Property 2061
Count 2061

QueryPrompt Class 2061
QueryPrompt Properties 2061
EditType 2061
FieldDecimal 2062
FieldFormat 2062
FieldLength 2063
FieldName 2063
FieldType 2063
HeadingText 2064
HeadingType 2064
LangCount 2065
Name 2065
PromptRecordFieldName 2065
PromptTable 2065
UniquePromptName 2066
UseCount 2066
Query Classes Examples 2066
Creating a New Query 2066
Adding Criteria 2068
Using Outer Joins 2074

Chapter 34

Record Class 2077

Understanding Record Class 2077
Shortcut Considerations 2078
Record Methods Used to Create SQL Statements 2078
Data Type of a Record Object 2079
Scope of a Record Object 2079
Record Class Built-in Functions 2079
Record Class Methods 2080
CompareFields 2080
CopyChangedFieldsTo 2081
CopyFieldsTo 2082
DBPatternMatch 2084
Delete 2085
ExecuteEdits 2086
GetField 2089
Insert 2090
Save 2092
SearchClear 2095
SelectByKey 2096
SelectByKeyEffDt 2098

SetDefault 2099
SetEditTable 2100
Update 2102
Record Class Properties 2104
FieldCount 2104
fieldname 2104
IsChanged 2105
IsDeleted 2105
IsEditError 2106
Name 2106
ParentRow 2107
RelLangRecName 2107
SystemIDFieldName 2108
TimeStampFieldName 2109

Chapter 35

Row Class 2111

Understanding Row Class 2111
Shortcut Considerations 2112
Data Type of a Row Object 2112
Scope of a Row Object 2112
Row Class Built-in Function 2112
Row Class Methods 2112
CopyTo 2113
GetNextEffRow 2114
GetPriorEffRow 2114
GetRecord 2115
GetRowset 2117
scrollname 2118
Row Class Properties 2118
ChildCount 2119
DeleteEnabled 2119
IsChanged 2120
IsDeleted 2121
IsEditError 2121
IsNew 2122
ParentRowset 2122
recname 2123
RecordCount 2123
RowNumber 2123
Selected 2124
Style 2124

Visible 2125

Chapter 36

Rowset Class 2127

Understanding Rowset Class 2127
Shortcut Considerations 2128
Data Type of a Rowset Object 2128
Scope of a Rowset Object 2128
Rowset Class Built-In Functions 2129
Rowset Class Methods 2129
ClearDeletesChanges 2129
CopyTo 2131
DeleteRow 2133
Fill 2135
FillAppend 2138
Flush 2140
FlushRow 2142
GetCurrEffRow 2143
GetFirstUserSortedRow 2143
GetLastUserSortedRow 2144
GetNewEffRow 2145
GetRow 2146
HideAllRows 2147
InsertRow 2148
IsUserSorted 2149
MapBufRowToUserSortRow 2150
MapUserSortRowToBufRow 2151
Refresh 2153
Select 2154
SelectNew 2156
SetDefault 2158
ShowAllRows 2159
Sort 2160
Rowset Class Properties 2161
ActiveRowCount 2161
ChangeOnInit 2162
DataAreaCollapsed 2162
DBRecordName 2163
DeleteEnabled 2163
EffDt 2164
EffSeq 2165
InsertEnabled 2165

IsEditError 2166
Level 2167
Name 2167
ParentRow 2168
ParentRowset 2168
RowCount 2168
SetComponentChanged 2169
TopRowNumber 2169

Chapter 37

RowsetCache Class 2171

Understanding a Rowset Cache 2171
Using the RowsetCache Class 2172
Error Handling 2172
Data Type of a RowsetCache Object 2173
Scope of a RowsetCache Object 2173
RowsetCache Class Built-in Functions 2173
RowsetCache Class Methods 2173
Delete 2173
Get 2174
Save 2175
RowsetCache Class Properties 2177
Content 2177
Description 2177

Chapter 38

Session Class 2179

Understanding Session Class 2179
Security and Access to the PeopleSoft System 2180
Error Handling 2180
Displaying Error Messages 2181
Regional Settings 2182
Trace Settings 2182
Session Object Declaration 2183
State Considerations 2183
Considerations for Declaring Collections 2184
Scope of a Session Object 2184
Session Class Built-in Function 2184
Session Class Methods 2184
Connect 2184

Disconnect 2186
API Instantiation Methods 2186
Session Class Properties 2187
ErrorPending 2188
PSMessages 2188
PSMessagesMode 2188
RegionalSettings 2189
Repository 2190
SuspendFormatting 2190
TraceSettings 2190
WarningPending 2190
Accessing a PSMessages Collection 2191
PSMessages Collection Methods 2191
DeleteAll 2191
DeleteItem 2192
First 2192
Item 2193
Next 2194
PSMessages Collection Property 2194
Count 2194
PSMessage Class Properties 2194
ExplainText 2194
MessageNumber 2195
MessageSetNumber 2195
MessageType 2195
Source 2196
Text 2197
Regional Settings Class Properties 2198
ClientTimeZone 2198
CurrencyFormat 2199
CurrencySymbol 2199
DateFormat 2199
DateSeparator 2200
DecimalSymbol 2200
DigitGroupingSymbol 2200
LanguageCD 2201
UseLocalTime 2202
1159Separator 2202
2359Separator 2203
Trace Setting Class Properties 2203
API 2204
COBOLStmtTimings 2204
ConnDisRollbackCommit 2204
DBSpecificCalls 2204
ManagerInfo 2205

NetworkServices 2205
NonSSBs 2205
OutputUNICODE 2205
PCExtFcnCalls 2206
PCFcnReturnValues 2206
PCFetchedValues 2206
PCIntFcnCalls 2206
PCListProgram 2207
PCParameterValues 2207
PCProgramStatements 2207
PCStack 2207
PCStartOfPrograms 2208
PCTraceProgram 2208
PCVariableAssignments 2208
RowFetch 2208
SQLStatement 2209
SQLStatementVariables 2209
SSBs 2209
SybBindInfo 2209
SybFetchInfo 2210
TraceFile 2210

Chapter 39

SOAPDoc Class 2211

Understanding theSOAPDoc Class 2211
SOAP Message Exchange Model 2212
SOAP Message Document 2212
The SOAPDoc Class 2214
SOAPDoc Object Creation 2214
SOAP Header Section 2214
SOAP Envelope Section 2215
SOAP Body Section 2215
SOAP Method Section 2216
SOAPDoc Section Access 2216
PeopleCode to Create a SOAPDoc 2217
SOAPDoc Fault Section 2217
The ValidateSOAPDoc Method 2217
SOAPDoc Objects and Messages 2218
Data Type of a SOAPDoc Object 2218
Scope of a SOAPDoc Object 2218
SOAPDoc Built-In Functions 2219
SOAPDoc Class Methods 2219

AddBody 2219
AddEnvelope 2220
AddFault 2222
AddHeader 2224
AddMethod 2225
AddParm 2227
GetParmName 2227
GetParmValue 2228
ValidateSOAPDoc 2229
SOAPDoc Class Properties 2230
BodyNode 2231
EnvelopeNode 2231
FaultCode 2231
FaultCodeS 2232
FaultString 2232
HeaderNode 2233
MethodName 2234
MethodNode 2234
ParmCount 2234
XmlDoc 2235
SOAPDoc Class Examples 2235
Creating a SOAP Document 2235
Reading an Existing SOAP Document 2236
Using SOAP XML Text 2236

Chapter 40

SQL Class 2239

Understanding SQL Class 2239
Using Record Class SQL 2241
Creating a SQL Definition 2241
Binding and Executing of SQL Statements 2241
Fetching From a Select 2243
Using Array of Any for Bind Values or Fetch Results 2243
Understanding Different "Styles" of SELECT 2243
Reusing a Cursor 2244
Using the ReuseCursor Property 2244
Reusing a Cursor Without the ReuseCursor Property 2245
Understanding SQL Objects and Application Engine Programs 2247
Declaring a SQL Object 2247
Scope of an SQL Object 2247
SQL Class Built-in Functions 2247
SQL Class Methods 2248

Close 2248
Execute 2249
Fetch 2250
Open 2252
SQL Class Properties 2254
BulkMode 2254
IsOpen 2255
LTrim 2256
ReuseCursor 2256
RowsAffected 2257
Status 2257
TraceName 2258
Value 2259

Chapter 41

SyncServer Class 2261

Understanding Synchronization Server Events and the SyncServer Class 2261
Understanding the Synchronization Events 2261
Understanding the SyncServer Class 2263
Considerations Using Synchronization Events and the SyncServer Class 2264
Using the Synchronization Server Events 2265
Using OnConflict 2265
Using OnSelect 2267
Using OnValidate 2268
Using Attachment Events 2269
Using Filtering 2270
Scope of SyncServer Objects 2271
SyncServer Class 2271
SyncServer Class Methods 2271
AddReference 2271
AddReferenceType 2272
CheckForChanges 2273
Notify 2275
SelectAll 2276
SyncServer Class Properties 2276
ClientPlatform 2277
ConflictAlgorithm 2277
ConflictStatus 2277
ExcludeProperty 2278
ExportLocation 2278
IsReferenceUnresolved 2279
LastSyncDateTime 2279

LastSyncRowCount 2279
PropertyValue 2280
SessionID 2280
Synchronizing 2281
SyncIDs 2281
TypeID 2282
validateID 2282
validateRowCount 2282
validateVersion 2283
Advanced Considerations for Synchronization 2283
Using OnValidateSet 2284
Using OnGetDefinition 2285

Chapter 42

TransformData Class 2287

Understanding the TransformData Class 2287
Creating a TransformData Object 2287
TransformData Class Properties 2288
DestMsgName 2288
DestMsgVersion 2288
DestNode 2288
RejectTransform 2288
RoutingDefnName 2289
SourceMsgName 2289
SourceMsgVersion 2289
SourceNode 2289
Status 2290
XmlDoc 2290

Chapter 43

Tree Classes 2291

Understanding Tree Classes 2291
Relationships Between Different Tree Classes 2292
Collections in the Tree Classes 2294
Error Handling With Trees 2294
Leaves and Nodes Insert Verification 2295
Restrictions on Trees When Used as a SmartNavigation Data Source 2296
Data Types for Tree Objects 2297
Scope of the Tree Objects 2297
Tree Classes Implementation 2297

Tree Classes Reference 2300
Session Class Methods 2300
GetTree 2301
GetTreeStructure 2301
Branch Collection 2302
Branch Collection Property 2302
Count 2302
Leaf Class 2302
Leaf Class Methods 2303
Cut 2303
Delete 2303
DeleteByRange 2304
GenABNMenuElement 2305
InsertDynSib 2305
InsertSib 2306
LoadABNChart 2307
LoadABNChartOrdered 2308
MoveAsChild 2309
MoveAsChildByName 2310
MoveAsSib 2311
MoveAsSibByRange 2312
PasteSib 2312
RefreshDescription 2313
UpdateRanges 2314
Leaf Class Properties 2314
Description 2315
DisplayLevelNumber 2315
Dynamic 2315
HasNextSib 2316
HasPrevSib 2316
ImageName 2316
IsChanged 2316
IsCut 2317
IsDeleted 2317
IsInserted 2317
NextSib 2317
Parent 2318
PrevSib 2318
RangeFrom 2318
RangeTo 2319
TreeBranchName 2319
TreeEffDt 2319
TreeName 2319
TreeSetId 2319
TreeUserKeyValue 2320

Level Collection 2320
Level Collection Methods 2320
    Add 2320
    Item 2321
    Remove 2321
Level Collection Properties 2322
    Count 2322
    First 2322
    Last 2322
    Next 2322
Level Class 2323
Level Class Methods 2323
    Create 2323
Level Class Properties 2323
    AllValuesAudit 2324
    Description 2324
    Name 2324
    Number 2324
    TreeBranchName 2325
    TreeEffDt 2325
    TreeName 2325
    TreeSetId 2325
    TreeUserKeyValue 2325
Node Class 2326
Node Class Methods 2326
    Branch 2326
    Cut 2327
    Delete 2327
    DeleteByName 2328
    Expand 2328
    GenABNMenuElement 2329
    GenBreadCrumbs 2330
    GenRelatedActions 2331
    InsertChildLeaf 2331
    InsertChildNode 2332
    InsertChildRecord 2333
    InsertDynChildLeaf 2334
    InsertSib 2335
    InsertSibRecord 2336
    LoadABNChart 2337
    LoadABNChartOrdered 2338
    MoveAsChild 2339
    MoveAsChildByName 2340
    MoveAsSib 2341
    MoveAsSibByName 2341

PasteChild 2342
PasteSib 2343
RefreshDescription 2344
Rename 2344
SwitchLevel 2345
Unbranch 2345
Node Class Properties 2346
AllChildCount 2346
AllChildNodeCount 2346
ChildLeafCount 2347
ChildNodeCount 2347
ColImageName 2347
Description 2347
DisplayLevelNumber 2347
ExpImageName 2348
FirstChildLeaf 2348
FirstChildNode 2348
HasChildLeaves 2349
HasChildNodes 2349
HasChildren 2349
HasNextSib 2349
HasPrevSib 2350
IsBranched 2350
IsChanged 2350
IsCut 2350
IsDeleted 2350
IsInserted 2351
IsRoot 2351
LastChildLeaf 2351
LastChildNode 2352
LevelNumber 2352
Name 2352
NextSib 2353
Parent 2353
PrevSib 2353
State 2354
TreeBranchName 2354
TreeEffDt 2354
TreeName 2355
TreeSetId 2355
TreeUserKeyValue 2355
Type 2355
Tree Class 2356
Tree Class Methods 2356
Audit 2356

AuditByName 2357
Close 2358
Copy 2359
Create 2360
Delete 2362
Exists 2363
FindLeaf 2364
FindNode 2365
FindRoot 2366
InsertRoot 2367
LeafExists 2368
LockTree 2369
Open 2369
OpenAsOfDate 2371
OpenForExport 2373
OpenWholeTree 2374
NodeExists 2375
Rename 2376
Save 2377
SaveAs 2378
SaveAsDraft 2379
SaveDraft 2381
SetImportMode 2382
TreeLocksNumber 2382
UnlockTree 2383
UpdateLock 2384
Tree Class Properties 2385
AllValues 2385
AuditDetails 2386
Branches 2386
BranchImageName 2386
BranchLevel 2386
BranchName 2387
Category 2387
Description 2387
DuplicateLeaves 2388
EffDt 2388
HasDetailRanges 2389
HasLockedBranches 2389
IsBranched 2389
IsChanged 2390
IsOpen 2390
IsQueryTree 2390
IsValid 2391
IsVersionChanged 2391

IsWholeTree 2391
KeyBranchName 2392
KeyEffDt 2392
KeyName 2392
KeySetId 2393
KeyUserKeyValue 2393
LeafCount 2393
LeafImageName 2393
LeafOnClipboard 2394
LevelCount 2394
Levels 2394
LevelUse 2395
LockOwner 2395
LockStatus 2396
Name 2396
NodeColImageName 2397
NodeCount 2397
NodeExpImageName 2397
NodeOnClipboard 2397
ParentLevel 2398
ParentName 2398
PerformanceMethod 2398
PerformanceSelector 2399
PerformanceSelectorOption 2399
SetID 2400
Status 2400
Structure 2401
StructureName 2401
TreeImageName 2402
UserKeyValue 2402
UseUpdateReservation 2402
Tree Structure Class 2402
Tree Structure Class Methods 2403
Close 2403
Copy 2403
Create 2404
Delete 2405
Open 2405
Rename 2406
Save 2406
Tree Structure Class Properties 2407
Description 2407
DetailComponent 2407
DetailField 2408
DetailMenu 2408

DetailMenuBar 2409
DetailMenuItem 2409
DetailMultiNavigate 2409
DetailPage 2410
DetailRecord 2410
IndirectionMethod 2411
KeyName 2412
LevelComponent 2412
LevelMenu 2412
LevelMenuBar 2413
LevelMenuItem 2413
LevelPage 2413
LevelRecord 2413
Name 2414
NodeComponent 2414
NodeField 2415
NodeMenu 2415
NodeMenuBar 2415
NodeMenuItem 2416
NodeMultiNavigate 2416
NodePage 2416
NodeRecord 2417
NodeUserKeyField 2417
SummarySetId 2417
SummaryLevelNumber 2418
SummaryTreeName 2418
SummaryUserKeyValue 2418
Type 2419
Traverse Tree Hierarchy Example 2419

Chapter 44

Universal Queue Classes 2421

Understanding Universal Queue Classes 2421
MCFFactory Class Hierarchy 2421
Scope of the Universal Queue Classes 2422
Data Types of the Universal Queue Classes 2422
How to Import Universal Queue Classes 2423
How to Create a Universal Queue Object 2424
Universal Queue Classes Built-in Functions 2424
Universal Queue Classes Constructors 2425
Agent 2425
AgentPhysQueueProps 2426

AgentPhysQueueTasks 2426
Broadcast 2427
LogicalQueue 2428
MCFFactory 2428
PhysicalQueue 2429
Task 2430
TaskList 2431
Util 2433
Agent Class 2433
Agent Methods 2434
Delete 2434
Refresh 2435
RefreshQTaskList 2435
Agent Properties 2436
AgentID 2436
AgentProps 2436
AgentTasks 2437
Buddy 2438
Language 2438
Name 2438
NickName 2438
PhysicalQueueID 2439
TotalPhysicalQueues 2439
AgentPhysQueueProps Class 2439
AgentPhysQueueProps Properties 2440
AgentID 2440
PhysicalQueueID 2440
SkillLevel 2440
WorkLoad 2441
AgentPhysQueueTasks Class 2441
AgentPhysQueueTasks Method 2441
Refresh 2441
AgentPhysQueueTasks Properties 2442
AcceptedTaskList 2442
AssignedTaskList 2442
AgentID 2443
PhysicalQueueID 2443
Broadcast Class 2443
Broadcast Class Method 2443
Broadcast 2444
LogicalQueue Class 2445
LogicalQueue Properties 2445
LogicalQueueID 2446
PhysicalQueueID 2446
MCFFactory Class 2446

MCFFactory Property 2446
LogicalQueue 2447
PhysicalQueue Class 2447
PhysicalQueue Methods 2447
Refresh 2447
RefreshTaskList 2448
PhysicalQueue Properties 2449
AcceptedTaskList 2449
AssignedTaskList 2449
Agent 2449
BrowserURL 2450
EnqueuedTaskList 2450
EscalatedTaskList 2450
InternalURL 2451
IsActive 2451
LogicalQueueID 2452
OverflowedTaskList 2452
PhysicalQueueID 2452
RENURLID 2452
TotalAgents 2453
Task Class 2453
Task Methods 2453
Close 2453
Enqueue 2454
Refresh 2456
RefreshStatus 2456
Task Properties 2457
AgentID 2457
ApplicationData 2457
Comments 2458
EnqueueTime 2458
EscalationTime 2458
Language 2458
OriginalTime 2458
OverFlowTime 2459
PhysicalQueueID 2459
TiedAgentID 2459
TaskList Class 2459
TaskList Method 2459
Refresh 2459
TaskList Properties 2460
AgentID 2460
PhysicalQueueID 2460
Task 2461
TaskType 2461

Total 2461
Util Class 2462
Util Methods 2462
GetLogicalQueue 2462
GetTimeDiff 2463
MCFFactory Example 2463

Chapter 45

Verity Search Classes 2465

Understanding Search Class 2465
Using the SearchResults Collection 2466
Understanding SearchResults 2467
Search Classes Hierarchy 2468
Error Handling 2470
Data Type of Search Objects 2471
Scope of the Search Objects 2472
Session Class Methods 2472
GetSearchIndexes 2472
GetSearchQuery 2473
PortalRegistry Class Methods 2473
BuildSearchIndex 2474
GetSearchQuery 2474
SearchQuery Class 2475
SearchQuery Class Methods 2475
Execute 2476
Parse 2476
SearchQuery Class Properties 2478
HitCount 2478
Indexes 2478
IndexName 2479
KnowledgeBase 2480
Language 2480
ProcessedCount 2480
QueryText 2480
RequestedFields 2480
ScorePrecision 2481
SortSpecifications 2481
ParseResult Collection 2482
ParseResult Collection Methods 2482
First 2482
Next 2482
ParseResult Collection Properties 2483

ErrorCount 2483
WarningCount 2484
ParseResult Class 2484
ParseResult Class Properties 2484
Message 2484
Severity 2484
SearchResult Collection 2485
SearchResult Collection Methods 2485
First 2485
Item 2485
Next 2486
SearchResult Collection Property 2486
Count 2487
SearchResult Class 2487
SearchResult Class Properties 2487
Key 2487
Score 2487
ScoreAsNumber 2488
SearchFields 2488
SearchField Collection 2488
SearchField Collection Methods 2489
First 2489
ItemByName 2489
Next 2490
SearchField Collection Property 2490
Count 2490
SearchField Class 2491
SearchField Class Properties 2491
Name 2491
Value 2491
SearchIndexes Collection 2491
SearchIndexes Collection Methods 2492
First 2492
ItemByName 2492
Next 2493
SearchIndex Collection Property 2493
Count 2493
SearchIndex Class 2493
SearchIndex Class Method 2494
Save 2494
SearchIndex Class Properties 2494
ExtraOptions 2495
FSOpts 2495
HTTPOpts 2495
Languages 2495

Location 2496
Name 2496
RecOpts 2496
Schedules 2496
Type 2497
SearchRecord Options Class 2497
SearchRecord Options Class Properties 2497
Fields 2498
Filter 2498
IncrementalView 2498
RecName 2498
VeggieKey 2499
ZoneOptions 2500
SearchRecordField Collection 2500
SearchRecordField Collection Methods 2500
DeleteItem 2501
First 2501
InsertItem 2502
ItemByName 2502
Next 2503
SearchRecordField Collection Property 2503
Count 2503
SearchRecordField Class 2504
SearchRecordField Class Properties 2504
FieldName 2504
IsAttachment 2504
IsVerityField 2505
IsWordIndex 2505
RecordName 2505
Search Language Collection 2505
Search Language Collection Methods 2505
DeleteItem 2506
First 2506
InsertItem 2507
ItemByName 2507
Next 2508
Search Language Collection Property 2508
Count 2508
Search Language Class 2509
Search Language Class Properties 2509
LanguageCd 2509
MapLanguageCd 2509
Search Schedule Collection 2509
Search Schedule Collection Methods 2510
DeleteItem 2510

First 2510
InsertItem 2511
ItemByName 2511
Next 2512
Search Schedule Collection Property 2512
Count 2512
Search Schedule Class 2513
Search Schedule Class Properties 2513
BuildType 2513
RunCntlId 2513
RunRecurrence 2514
ServerName 2514
Search HTTP Options Class 2514
Search HTTP Options Class Properties 2514
AllowHTTPS 2515
DomainLimit 2515
GlobList 2515
GlobListType 2515
LinkDepth 2516
MIMEList 2516
MIMEListType 2517
ProxyHost 2517
ProxyPort 2517
StartOpts 2518
Search FS Options Class 2518
Search FS Options Class Properties 2518
GlobList 2518
GlobListType 2519
MIMEList 2519
MIMEListType 2520
StartOpts 2520
Search Start Options Collection 2520
Search Start Options Collection Methods 2521
DeleteItem 2521
First 2521
InsertItem 2522
ItemByName 2522
Next 2523
Search Start Options Collection Property 2523
Count 2523
Search Start Options Class 2524
Search Start Options Class Properties 2524
IsDomainRestricted 2524
IsHostRestricted 2524
Value 2525

Search API Examples 2525
General Purpose Search API Example 2525
Portal Search API Example 2526

Chapter 46

XmlDoc Classes 2529

Understanding XmlDoc Classes 2529
When to Use an XmlDoc Object 2530
XmlDoc Object Creation 2531
Considerations Using a Unique Namespace 2531
Considerations Using Rowsets 2532
XmlNode Class Considerations 2533
Accessing and Traversing an XmlNode Object 2534
Error Handling 2534
SOAPDoc Object Considerations 2534
Scope of XmlDoc and XmlNode Objects 2535
Data Type of an XmlDoc or XmlNode Object 2535
XmlDoc Classes Built-in Functions 2536
XmlDoc Class Methods 2536
CopyRowset 2536
CopyToPSFTMessage 2537
CopyToRowset 2539
CreateDocumentElement 2541
CreateDocumentType 2542
GenFormattedXmlString 2544
GenXmlFile 2545
GenXmlString 2546
GetElementsByTagName 2546
LoadIBContent 2548
ParseXmlFromURL 2549
ParseXmlString 2551
XmlDoc Properties 2552
DocumentElement 2552
IsNull 2552
XmlNode Class 2553
XmlNode Class Methods 2553
AddAttribute 2553
AddAttributeNS 2554
AddCDataSection 2555
AddComment 2557
AddElement 2558
AddElementNS 2559

AddEntityReference 2560
AddNode 2561
AddProcessInstruction 2562
AddText 2563
CopyNode 2564
FindNode 2565
FindNodes 2566
GenXmlString 2567
GetAttributeName 2568
GetAttributeValue 2569
GetCDataValue 2569
GetCDataValues 2570
GetChildNode 2571
GetElement 2572
GetElements 2573
GetElementsByTagName 2574
GetElementsByTagNameNS 2575
InsertCDataSection 2576
InsertComment 2577
InsertElement 2579
InsertElementNS 2580
InsertEntityReference 2581
InsertNode 2582
InsertProcessInstruction 2584
InsertText 2585
RemoveAllChildNode 2586
RemoveChildNode 2587
XmlNode Class Properties 2589
AttributesCount 2589
ChildNodeCount 2589
Index 2590
IsNull 2590
LocalName 2590
NamespaceURI 2591
NextSibling 2591
NodeName 2591
NodePath 2591
NodeType 2592
NodeValue 2592
ParentNode 2593
Prefix 2593
PreviousSibling 2593

Chapter 47

XML Publisher Classes **2595**

Understanding XML Publisher and the XML Publisher Classes 2595
XML Publisher Terms 2596
XML Publisher Flow Diagram 2597
Error Handling 2597
Data Type and Scope of XML Publisher Objects 2598
Instantiating XML Publisher Objects 2598
Search Operator Values 2599
XML Publisher Classes Reference 2599
XML Publisher Report Manager Definition Classes 2600
ReportDefn Class 2600
ReportDefn Class Constructor 2600
    ReportDefn 2600
ReportDefn Class Methods 2601
    Close 2601
    DisplayOutput 2602
    Get 2602
    GetOutDestFormatString 2603
    GetPSQueryPromptRecord 2604
    PrintOutput 2604
    ProcessReport 2605
    Publish 2606
    SetPSQueryPromptRecord 2607
    SetRuntimeDataXMLFile 2608
    SetRuntimeProperties 2609
ReportDefn Class Properties 2611
    Description 2611
    DestinationPrinter 2611
    FolderName 2611
    OutDestination 2611
    OutDestinationType 2612
    ProcessInstance 2612
    Status 2613
    UseBurstValueAsOutputFileName 2613
XML Publisher Report Manager Search Classes 2613
Report Class 2613
Report Class Constructor 2614
    Report 2614
Report Class Properties 2615
    contentId 2615

CreatedDateTime 2615
DatabaseName 2615
Description 2616
ExpireDate 2616
FileURL 2616
FolderName 2616
ProcessInstanceID 2617
ReportInstanceID 2617
ReportName 2617
ReportURL 2617
ReportManager Class 2618
ReportManager Class Constructor 2618
ReportManager 2618
ReportManager Class Methods 2619
AddSearchFieldCriteria 2619
GetReportList 2620
SetBurstFieldCriteria 2620
SetCaseSensitive 2621
SetDateCriteria 2622
SetFolderCriteria 2623
SetProcessInstanceCriteria 2623
SetReportIDCriteria 2624
SetUserIdCriteria 2624
SearchAttribute Class 2625
SearchAttribute Class Constructor 2625
SearchAttribute 2625
SearchAttribute Class Properties 2626
attrName 2626
attrValue 2626
compareOp 2626
XML Publisher Engine Classes 2627
PageNumber Class 2627
PageNumber Class Constructor 2627
PageNumber 2627
PageNumber Class Properties 2628
BackgroundFile 2628
FontName 2628
FontSize 2629
PositionX 2630
PositionY 2630
StartFromPageNum 2630
StartNum 2631
PDFMerger Class 2631
PDFMerger Class Constructor 2631
PDFMerger 2631

PDFMerger Class Method 2632
MergePDFs 2632
PDFMerger Class Properties 2633
ConfProp 2633
Locale 2633
PageNumber 2634
Watermark 2634
Properties Class 2635
Properties Class Constructor 2635
Properties 2635
Properties Class Methods 2636
GetProperty 2636
SetProperty 2637
Watermark Class 2638
Watermark Class Constructor 2638
Watermark 2639
Watermark Class Properties 2639
ImageFile 2639
ImageFileLowerLeftX 2640
ImageFileLowerLeftY 2640
ImageFileUpperRightX 2640
ImageFileUpperRightY 2640
PageIndex 2641
Text 2641
TextAngle 2641
TextFontName 2641
TextFontSize 2641
TextStartPosX 2642
TextStartPosY 2642
XML Publisher Classes Example 2642
Generating and Publishing a Report 2642

Appendix A

Quick Reference for PeopleCode Classes 2645

PeopleCode Syntax Quick Reference 2645
PeopleCode Classes Alphabetical Reference 2649
Typographical Conventions and Visual Cues 2649
AESection 2650
Analytic Calculation Engine Classes 2651
Analytic Calculation Engine Metadata Classes 2653
Analytic Grid Classes 2663
Analytic Type Classes 2664

- Application Classes 2666
- Array 2667
- BPEL 2669
- Charting Classes 2670
- Component Interface 2681
- Connected Query Classes 2681
- Crypt Class 2686
- Document Classes 2687
- Exception Class 2692
- Feed Classes 2693
- Field 2713
- File 2718
- Grid Classes 2720
- Internet Script Classes 2721
- Java 2725
- Mail Classes 2725
- MCF IM Classes 2735
- Message Classes 2737
- Notification Classes 2747
- Optimization PeopleCode Classes 2752
- Page 2756
- PortalRegistry 2757
- PostReport 2757
- Process Request Classes 2758
- Query 2762
- Record 2762
- Row 2763
- Rowset 2765
- RowsetCache 2768
- Search 2769
- Session 2769
- SOAPDoc 2769
- SQL 2770
- TransformData 2772
- Tree 2773
- Universal Queue 2773
- XmlDoc Classes 2778
- XML Publisher 2782

Session Classes 2787
- Session Classes Methods and Properties 2787
- API Repository Classes Methods and Properties 2793
- Component Interface Class Methods and Properties 2796
- Verity Search Classes Methods and Properties 2800
- PortalRegistry Classes Methods and Properties 2809
- Query Classes Methods and Properties 2837

Tree Classes Methods and Properties 2857
Deprecated Items and PeopleCode No Longer Supported 2869
Mapping of Old Objects to New Objects 2869
Deprecated Products and Classes 2871
Deprecated Functions 2881
Deprecated Methods and Properties 2885
Deprecated XML Publisher Items 2886
XML Publisher Items No Longer Supported 2887
Deprecated Messaging PeopleCode Functions, Methods and Properties 2895
Functions No Longer Supported 2898

Index 2901

PeopleCode API Reference Preface

This preface introduces the PeopleCode API Reference PeopleBook.

PeopleCode API Reference

PeopleCode is the proprietary language used in the development of Oracle's PeopleSoft applications. This PeopleBook provides a reference for the PeopleCode application programming interface (API)—that is, the classes delivered with PeopleCode. The chapters of this PeopleBook describe the syntax and fundamental elements of this part of the PeopleCode language—for example, all of the methods and properties of the Rowset class, the Field class, and so on.

There are two accompanying PeopleBooks about PeopleCode: the *PeopleCode Language Reference* and the *PeopleCode Developer's Guide.* The language reference covers the language elements of PeopleCode, such as the built-in functions, meta-SQL, system variables, and so on. Its chapters contain reference material for the PeopleCode language. The *PeopleCode Developer's Guide* contains conceptual information about developing programs with the PeopleCode language, the PeopleCode API, the component processor, and so on.

PeopleBooks and the Online PeopleSoft Library contains general information about Oracle's PeopleTools product line, such as related documentation, common page elements, and typographical conventions.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide

PeopleCode Typographical Conventions

Throughout this book, we use typographical conventions to distinguish between different elements of the PeopleCode language, such as bold to indicate function names, italics for arguments, and so on.

Please take a moment to review the following typographical cues:

Font Type	*Description*
`monospace font`	Indicates a PeopleCode program or other example
Keyword	In PeopleCode syntax, items in **keyword** font indicate function names, method names, language constructs, and PeopleCode reserved words that must be included literally in the function call.

Font Type	***Description***
Variable	In PeopleCode syntax, items in *variable* font are placeholders for arguments that your program must supply.
...	In PeopleCode syntax, ellipses indicate that the preceding item or series can be repeated any number of times.
{*Option1*|*Option2*}	In PeopleCode syntax, when there is a choice between two options, the options are enclosed in curly braces and separated by a pipe.
[ ]	In PeopleCode syntax optional items are enclosed in square brackets.
&Parameter	In PeopleCode syntax an ampersand before a parameter indicates that the parameter is an already instantiated object.

PeopleBooks and the PeopleSoft Online Library

A companion PeopleBook called *PeopleBooks and the PeopleSoft Online Library* contains general information, including:

- Understanding the PeopleSoft online library and related documentation.
- How to send PeopleSoft documentation comments and suggestions to Oracle.
- How to access hosted PeopleBooks, downloadable HTML PeopleBooks, and downloadable PDF PeopleBooks as well as documentation updates.
- Understanding PeopleBook structure.
- Typographical conventions and visual cues used in PeopleBooks.
- ISO country codes and currency codes.
- PeopleBooks that are common across multiple applications.
- Common elements used in PeopleBooks.
- Navigating the PeopleBooks interface and searching the PeopleSoft online library.
- Displaying and printing screen shots and graphics in PeopleBooks.
- How to manage the locally installed PeopleSoft online library, including web site folders.

- Understanding documentation integration and how to integrate customized documentation into the library.
- Application abbreviations found in application fields.

You can find *PeopleBooks and the PeopleSoft Online Library* in the online PeopleBooks Library for your PeopleTools release.

Chapter 1

AESection Class

This chapter provides an overview of AESection class and discusses the following topics:

- How an AESection is accessed.
- AESection example.
- Data type of an AESection object.
- Scope of an AESection object.
- AESection Class built-in function.
- AESection Class reference.

Understanding AESection Class

Before PeopleTools 8, users could perform SQL directly on the Application Engine tables, thereby changing their SQL and Application Engine "flow" in a dynamic manner, prior to running their applications. Some applications, for example, let the user input their "rules" in a user-friendly application, then convert these rules, at save time, into Application Engine constructs.

With PeopleTools 8, Application Engine programs should not perform SQL directly on the Application Engine tables, as they are system tables, and are cached. SQL on the Application Engine tables may not be accurately reflected when the applications are executed, because of the caching mechanism.

To overcome this problem, developers have two basic operations that let them modify their Application Engine programs from their online pages:

- Ability to modify SQL definitions, which are referenced within Application Engine SQL using the PeopleCode SQL class and the meta-SQL function %SQL.
- Ability to dynamically change the execution flow of a given Application Engine section.

Use the AESection class to do the latter. This section object is used to modify the steps and SQL associated with a given section by using PeopleCode.

Before describing the specifics of the AESection class, the following are some general terms to understand:

- *base* section

  The *base* section represents the Application Engine section you are working on. This is the section that you are changing in PeopleCode. In other words, it's the target section.

- *template* section

  The *template* section represents the Application Engine section that is the source, or "model" for the section you're building. You copy *from* the template section *to* the base section.

The template must exist in the database before you can use it.

If the base section you specify doesn't exist, a *new* base section is created. This enables you to dynamically create sections as needed.

You can copy steps (and their attributes) from the template to the base. The only attribute of the step you can modify is the SQL statement that gets executed.

Warning! When you open or get an AESection object, (that is, the base section) any existing steps in the section are deleted. You must add a new step to the section before you can modify it.

The main assumption for this class is that your rules are dynamic primarily in the SQL that they execute, but that the structure of the rules are static, or at least defined well enough that a standard template can be applied.

Note that you can't update the Application Engine *actions* that are not SQL-related (that is, PeopleCode, Call Section, or Log Message). In other words, you can't change the PeopleCode associated with a PeopleCode action within a step. You can add a step containing a PeopleCode action to your new section, but you can't change the PeopleCode dynamically.

See Also

Chapter 40, "SQL Class," page 2239

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "Meta-SQL Elements," %SQL

Enterprise PeopleTools 8.51 PeopleBook: Application Engine, "Creating Application Engine Programs," Adding Steps

How an AESection is Accessed

When an AESection is opened (or accessed), the system first checks if it exists with the given input parameters.

If the base section you specify doesn't exist, a *new* base section is created. This enables you to dynamically create sections as needed. In addition, if the target section doesn't exist, all section-level attributes are copied from the template to the target section. If the target exists, it retains its attribute settings.

If an effective date is specified (with EffDt), but there is no match using that effective date, the AESection is opened using the base effective date of '1900-01-01'.

The market defined for the current component is used for the section to be open. If no section with this market exists, the default GBL is used.

If you open an AESection object from within a running Application Engine program, the market value is set to the value of the current process.

The database platform you're currently running is used as the database platform for the section to be opened. If no section with this database platform exists, the default "none" is used.

To find the correct section to open, the precedence is:

1. Market
2. Database platform
3. Effective date

When a section is closed and its changes are saved, the market and database platform from the system that performed the changes is used as the market and database platform for the modified section.

Let's take an example. Assume that you're running on DB2 UDB for OS/390 and z/OS, and your current market is set to 'MKT'. Assume that you want to open a section called Sect1 in an Application Engine program called TestAppl, and that your data looks something like this:

Number	***Application Engine Program***	***Application Engine Section***	***Market***	***Platform***	***Effective Date***
1	TestAppl	Sect1	USA	DB2	1990-01-01
2	TestAppl	Sect1	USA	None	1900-01-01
3	TestAppl	Sect1	GBL	None	1900-01-01

If you use:

```
&SECTION.Open("TestAppl", "Sect1", "1998-12-14");
```

Then the third section in the previous list is used, because the market takes highest precedence. When the section is saved, the values for market and platform are updated to the current system values.

Warning! When you open or get an AESection object, (that is, any base section) any existing steps in the section are deleted. You must add a new step to the section before you can modify it.

The other attributes of the section, however, are retained, with the exceptions noted previously.

AESection Example

Assume that you have a template section called TEMPLATE in the Application Engine program called MY_APPL. The template looks like this:

Steps	***Actions***
NewStep1	DO When DO Select SQL
NewStep2	DO Select CallSection

Also, assume that you have a base section called DYN_SECT in the Application Engine program called RULES. When you start, this section looks like this:

Steps	***Actions***
Step1	DO Select Call Section DO Unit
Step2	Call Section

Here's the PeopleCode:

```
Local AESection &Section;
&Section = GetAESection("RULES", "DYN_SECT");
/* Open the base section */
&Section.SetTemplate("MY_APPL", "TEMPLATE");
/* Set the template section */
&Section.AddStep("NewStep2");
/* Insert NewStep2     */
/* Do some SQL stuff here  */
&Section.SetSQL("DO_SELECT", &MySql);
/* Modify the SQL in the added step */
&Section.Save();
&Section.Close();
/* Save and close   */
```

The base section looks like this after execution:

Steps	***Actions***
NewStep2	DO Select Call Section

Note. The existing steps in the base section have been *overwritten* by the new step from the template section.

Data Type of an AESection Object

You should declare an AESection object as type AESection. For example:

```
Local AESection &SECTION;
```

Scope of an AESection Object

An AESection object can only be instantiated from PeopleCode.

The AESection Object is designed for use within online pages. Typically, dynamic sections should be constructed in response to an end-user action.

Note. Do not call an AESection object from an Application Engine PeopleCode Action. If you need to access another section, use CallSection Actions instead.

See Also

Enterprise PeopleTools 8.51 PeopleBook: Application Engine, "Creating Application Engine Programs," Specifying Call Section Actions

AESection Class Built-in Function

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAESection

AESection Class Methods

In this section, we discuss each AESection class method, in alphabetical order.

AddStep

Syntax

```
AddStep(ae_step_name [, NewStepName])
```


Description

The given step name from the template section is added as the next step into the base section, and named the existing step name.

Note. When you open or get a section, all the existing steps are deleted. The first time you execute AddStep, you add the first step. The second time you execute AddStep, you add the second step, and so on.

All attributes of the step are copied, including all of its actions. The only changeable attribute of a step are its SQL statements, which can be modified using the SetSQL method. SetSQL is run on the *current* step, that is, the step most recently added.

You can also change the name of the step by using the optional parameter *NewStepName.*

If the step named does not exist in the template, an error occurs. Likewise, if you have not opened or set the template for the section, you receive an error.

Parameters

Parameter	***Description***
ae_step_name	Specify the step name from the template to be added as the *next* step in the base section. This parameter takes a string value.
NewStepName	Specify a new name for the step to be added. This parameter is optional, and takes a string value.

Returns

None.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," SetSQL, page 9

Close

Syntax

```
Close()
```


Description

Close closes the AESection object. Any unsaved changes to the section are discarded.

Parameters

None.

Returns

None.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," Save, page 8 and Chapter 1, "AESection Class," Open, page 7

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAESection

Open

Syntax

Open*(ae_applid,ae_section,* [*effdt*])

Description

The Open method associates the AESection object with the given Application Engine section, based on the *ae_applid* and *ae_section*. If the *effdt* is specified, this is also used to get the section object. In other words, the Open method sets the base section.

Warning! When you open or get an AESection object, (that is, the base section) any existing steps in the section are deleted.

Note. If the base section you specify doesn't exist, a *new* base section is created. This enables you to dynamically create sections as needed. In addition, if the target section doesn't exist, all section-level attributes are copied from the template to the target section. If the target exists, it retains its attribute settings.

If the AESection is still open when you issue the Open method, the previously opened object is discarded and none of the changes saved. To prevent accidentally discarding your changes, you can use the IsOpen property to verify if a section is already open.

The AESection is open based on the Market and database type of your current system.

Parameters

Parameter	***Description***
ae_applid	Specify the application ID of the section you want to access. This parameter takes a string value.
ae_section	Specify the section name of the section you want to access. This parameter takes a string value.
effdt	Specify the effective date of the section you want to access (optional). This parameter takes a string value.

Returns

An AESection object.

Example

```
Local AESection &SECTION;

&SECTION = GetAESection("RULES1", "DYN_SECT");

/* do some processing */

&SECTION.Close();

&SECTION.Open("RULES2", "DYN_SECT");

/* do additional processing */
```

See Also

Chapter 1, "AESection Class," IsOpen, page 11

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAESection

Save

Syntax

```
Save()
```


Description

The Save method saves the section to the database. The AESection object remains open after you use this method. To close the object, you must use the Close method.

You must commit all database changes prior to using this method. This is to avoid locking critical Tools tables and hence freezing all other users. You receive a runtime error message if you try to use this method when there are pending database updates, and your PeopleCode program terminates. You need to commit any database updates prior to using this method. Use the CommitWork function to commit database updates.

Parameters

None.

Returns

None.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," Close, page 6 and Chapter 1, "AESection Class," Open, page 7

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CommitWork

SetSQL

Syntax

SetSQL*(action_type_string, string)*

Description

The SetSQL method replaces the SQL associated with the given action type in the current step in the base section with the SQL in *string.* The *current* step is the latest step that was added using AddStep. The action types are:

- DO_WHEN
- DO_WHILE
- DO_SELECT

- SQL
- DO_UNTIL

Note. All action types must be passed in as strings with quotation marks.

If the action specified does not exist in the current step, an error occurs.

You can use a SQL object as *string*.

```
&SECTION.SetSQL("SQL", &SQL);
```

Parameters

Parameter	***Description***
action_type	Specifies the action type of the current step that should be changed. This parameter takes a string value.
string	Specifies the SQL to be used to replace the SQL in the current step.

Returns

None.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," AddStep, page 5

Chapter 40, "SQL Class," page 2239

SetTemplate

Syntax

`SetTemplate``(`*`ae_applid,ae_section`*`)`

Description

The SetTemplate method sets the template to be used with an AESection object, as identified by *ae_applid* and *ae_section.*

The rules for assigning a template section are similar to the rules for selecting a base section. The selection of market and database platform are done exactly the same. However, the effective date of the template is always set based on the effective date of '1900-01-01'.

You must set the template before you can use any of the other methods.

Parameters

Parameter	***Description***
ae_applid	Specify the name of the Application Engine program that contains the section to be used as the template.
ae_section	Specify the name of the Application Engine section in the program to be used as the section template.

Returns

None.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," Open, page 7

Chapter 1, "AESection Class," How an AESection is Accessed, page 2

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAESection

AESection Class Property

In this section, we discuss the IsOpen AESection property.

IsOpen

Description

If this property is True, the section object is already open. If you try to open a section object that is already open, the open section object is closed and all the changes that haven't been saved are discarded before the object is re-opened.

Example

```
If Not(&MYSECTION.IsOpen) Then
   &MYSECTION.Open(MYAPPLID, SECTION2);
End-If;
```

Chapter 2

Analytic Calculation Engine Classes

This chapter provides an overview of the Analytic Calculation Engine classes, and discuses:

- Using the Analytic Calculation Engine classes with Application Engine
- Running synchronously
- Error handling
- Data types for the Analytic Calculation Engine classes
- Scope of the Analytic Calculation Engine class objects
- Analytic Calculation Engine reference

Important! The Analytic Calculation Engine classes are not supported on IBM z/OS and Linux for IBM System z platforms.

Understanding the Analytic Calculation Engine Classes

PeopleSoft Analytic Calculation Engine comprises a calculation engine plus several PeopleTools features which enable application developers to define both the calculation rules and the display of calculated data within PeopleSoft applications for the purposes of multi-dimensional reporting, data editing and analysis.

More specifically, developers create *analytic models* in order to define the rules which are used to calculate data. Developers also create PeopleSoft Pure Internet Architecture pages with *analytic grids* in order to display the data within PeopleSoft applications. End users view, analyze and make changes to this data. When end users save their changes, PeopleSoft Analytic Calculation Engine recalculates the data and saves the calculated data to the database.

PeopleCode enables developers to manipulate analytic calculation data as follows:

- Use the Analytic Calculation Engine classes to either retrieve or specify data in an instance of an analytic model loaded into the system, and also to calculate (or recalculate) cube values.
- Use the Analytic Calculation Engine metadata classes to manipulate an analytic model definition. For example, you can add cubes to a cube collection or rename an existing user function for a model.
- Use the Analytic Grid classes to manipulate the display of analytic calculation data on a page.
- Use the Analytic Type classes to manipulate an analytic type definition. For example, you can specify a new analytic model for an analytic type definition.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," page 53

Chapter 4, "Analytic Grid Classes," page 159

Chapter 5, "Analytic Type Classes," page 169

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine

Using the Analytic Calculation Engine Classes with Application Engine

All Application Engine programs run on the application engine server. This means your program runs on the application engine server, not the analytic server.

PeopleSoft recommends that you put each of the following methods into its own step when you're creating an analytic calculation using a Application Engine program:

- AnalyticInstance Load method
- AnalyticInstance Unload method
- AnalyticInstance Copy method
- AnalyticInstance Delete method (in case of any readable/writable records in the analytic type)
- AnalyticModel Recalculate method
- CubeCollection SetData method

Also, with the Recalculate method, you should use the Commit after step option.

Running Synchronously

Some of the methods and functions associated with these classes have the option of being run asynchronously. When you run these asynchronously, and not from a Application Engine program, these methods and functions have additional functionality, that is, if the RenServer is configured, a window displays the PeopleSoft Analytic Calculation Engine execution status.

Using Trees

PeopleSoft Analytic Calculation Engine uses trees to establish hierarchies of a dimension's parent-child relationships. PeopleSoft Analytic Calculation Engine uses these hierarchies to:

- Calculate and display aggregated data to end users.
- Enable end users to navigate through data by performing such actions as expanding and collapsing nodes.

- Enable end users to drill down and drill up through data.

It is important to understand that PeopleSoft trees and hierarchies differ in the following manner: You create one tree for each dimension that requires a hierarchy. The analytic model uses that tree to create one hierarchy for one dimension.

Use the AttachTree method to attach a tree to a dimension, and use the DetachTree method to detach the tree from the dimension.

You should be aware of the following restrictions:

- Because the AttachTree method attaches a specific tree to an analytic instance, the system throws an error if the tree's name, setID or effective date is incorrect.
- You can attach only one tree to one dimension at a time.
- If the analytic instance is already loaded into the analytic server, the tree isn't attached until the analytic model is loaded.
- No aggregate data is displayed to the user after you use the DetachTree method.
- If your application loads the analytic model after the tree has been detached, the analytic model doesn't create a hierarchy for the dimension.
- If the analytic instance is already loaded into the analytic server, the DetachTree method isn't applied to the tree until the next time your application loads the analytic instance.

You can only attach a tree to an analytic model before the analytic instance

No aggregate data displayed to user after detach tree.

See Also

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Creating Hierarchies"

Error Handling

All the Analytic Calculation Engine classes throw PeopleCode exceptions for any fatal error that occurs in the execution of the operation. PeopleSoft recommends enclosing your analytic model programs in try-catch statements. This way, if your program catches the exception, the message set and message number that are associated with the exception object indicate the error.

Using the Messages Property

Use the Messages property to determine whether any errors occurred. The Messages property returns a multi-dimensional array. You can use the Len property of the array class to determine the length of the array. If the array has a length of 0, there are no errors.

Data Types for Analytic Calculation Engine Classes

Every PeopleSoft Analytic Calculation Engine object is declared as its own data type, that is, AnalyticModel objects are declared as type AnalyticModel, CubeCollection objects are declared as type CubeCollection, and so on.

The following are the data types for the PeopleSoft Analytic Calculation Engine classes:

- AnalyticInstance
- AnalyticModel
- CubeCollection

Scope of Analytic Calculation Engine Classes

The Analytic Calculation Engine objects can only be instantiated from PeopleCode.

These objects can be used anywhere you have PeopleCode, that is, in a Application Engine program, an application class, record field PeopleCode, and so on.

Analytic Calculation Engine objects can be of scope Local, Component, or Global.

Analytic Calculation Engine Classes Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateAnalyticInstance

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAnalyticInstance

AnalyticInstance Class Methods

In this section, we discuss the AnalyticInstance methods. The methods are discussed in alphabetical order.

CheckAsyncStatus

Syntax

`CheckAsyncStatus`*`(OperationID)`*

Description

Use the CheckAsyncStatus method to determine the current state of the asynchronous methods, as well as the Load method when executed asynchronously.

Parameters

Parameter	***Description***
OperationID	Specify the operation ID of the asynchronous operation. This operation ID is returned when one of the asynchronous operations is executed.

Returns

This method returns one of the following values:

Value	***Description***
Less than 0	The operation completed with errors.
0	The operation has not completed yet.
Greater than 0	The operation completed successfully.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CheckStatus, page 17; Chapter 2, "Analytic Calculation Engine Classes," Load, page 21 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 24

CheckStatus

Syntax

```
CheckStatus()
```

Description

Use the CheckStatus method to determine the current state of the analytic calculation engine.

Parameters

None.

Returns

This method returns one of the following values:

Value	*Description*
%AnalyticInstance_NotLoaded	The analytic instance has not been loaded into the application server.
%AnalyticInstance_Loading	The analytic instance is being loaded.
%AnalyticInstance_Idle	The analytic instance is idle, that is, not currently running.
%AnalyticInstance_Busy	The analytic instance is currently running.
%AnalyticInstance_ Inaccessible	The analytic instance is currently inaccessible.
%AnalyticInstance_ Terminating	The analytic instance is currently ending.

See Also

Chapter 2, "Analytic Calculation Engine Classes," Load, page 21 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 24

Copy

Syntax

`Copy``(`*`NewID, ForceDelete&Record`*`)`

Description

Use the Copy method to copy all the data and metadata to a new analytic instance from the current analytic instance.

If the specified *NewID* exists, or any data for *NewID* exists, the data is not copied unless *ForceDelete* has been specified as true.

If a tree is attached to the existing analytic instance, all tree data is also copied to the new analytic instance.

In addition, the analytic instance and the tree information are not copied unless the record specified with the *&Record* parameter is populated with the existing analytic instance ID.

Parameters

Parameter	*Description*
NewID	Specify the new analytic instance ID as a string.
ForceDelete	If the specified *NewID* exists, and *ForceDelete* is specified as false, the copy is cancelled. If the specified *NewID* exists, and *ForceDelete* is specified as true, the analytic instance specified by *NewID* is deleted, and a new analytic instance is created from the current ID.
&Record	Specify an already instantiated record object to pass in values to the Copy application package method that's defined with the specified analytic type definition.

Returns

None.

Example

```
&ai.Copy("MYTESTCOPY", True, &MyRecord);
```

See Also

Chapter 2, "Analytic Calculation Engine Classes," Delete, page 19; Chapter 2, "Analytic Calculation Engine Classes," Load, page 21 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 24

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions"

Delete

Syntax

Delete*(ForceUnload,&RecordRef)*

Description

Use the Delete method to remove all the metadata related to the given analytic instance.

Every analytic type definition is defined with an application package that contains three methods: Create, Delete, and Copy. The values in *&RecordRef* are passed to the Delete method.

This method does not delete the analytic instance if the analytic instance is currently loaded. Specify *ForceUnload* as true to unload the analytic instance before deleting it.

Parameters

Parameter	***Description***
ForceUnload	Specify whether to force the deletion of the analytic instance. This parameter takes a Boolean value. If the analytic instance is currently loaded, specifying true for this parameter forces the analytic instance to be unloaded before it is deleted.
&RecordRef	Specify a record to pass in values to the application package Delete class that's associated with the analytic type definition.

Returns

None.

Example

```
&ai.Delete(TRUE, &Record);
```

See Also

Chapter 2, "Analytic Calculation Engine Classes," Unload, page 24

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions"

GetAnalyticModel

Syntax

`GetAnalyticModel``(`**`Model.`***`ModelName`*`)`

Description

Use the GetAnalyticModel method to return a reference to an AnalyticModel object.

Parameters

Parameter	Description
ModelName	Specify the name of the analytic model for which you want to return a reference. The model definition must already exist in Application Designer.

Returns

A reference to an AnalyticModel object or Null.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AnalyticModel Class Methods, page 26

Load

Syntax

Load*(Sync,IdleTimeOut,***Message.***messagename)*

Description

Use the Load method to load the AnalyticInstance object executing the method into the analytic server.

If this analytic instance is already loaded, this method fails.

Parameters

Parameter	Description
Sync	Specify whether the analytic instance should be run synchronously or asynchronously. This parameter takes a Boolean value: true if it should be run synchronously, false otherwise. See Chapter 2, "Analytic Calculation Engine Classes," Running Synchronously, page 14.
IdleTimeOut	Specify, in minutes, the idle time out value. A value of 0 is an infinite time out. A value of -1 indicates that it should use the value specified in the default configuration for the analytic server. **Note.** When loading an analytic grid, you should always specify a value of -1. See *Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine*, "Managing Analytic Servers," Loading and Unloading Analytic Instances.

Parameter	Description
Message.*MessageName*	Specify an application message that should be sent if the analytic server crashes while the analytic instance is loaded. **Note.** The message is sent when the analytic server process restarts itself after crashing.

Returns

A string. For asynchronous loads, this string is passed to the CheckAsyncStatus method.

This method returns one of the following values:

Value	Description
%AnalyticInstance_NotLoaded	The analytic instance has not been loaded into the analytic server.
%AnalyticInstance_Loading	The analytic instance is being loaded.
%AnalyticInstance_Idle	The analytic instance is idle, that is, not currently running.
%AnalyticInstance_Busy	The analytic instance is currently running.
%AnalyticInstance_ Inaccessible	The analytic instance is currently inaccessible.
%AnalyticInstance_ Terminating	The analytic instance is currently terminating.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CheckAsyncStatus, page 16 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 24

RunAsync

Syntax

`RunAsync``()`

Description

Use the RunAsync method to specify if the analytic instance executing the method should be run in an asynchronous manner.

This method is used only to define the transaction in the analytic type definition in optimization. PeopleSoft Analytic Calculation Engine programs uses the Load method for defining synchronous or asynchronous operation.

Parameters

None.

Returns

None.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions

RunSync

Syntax

`RunSync``()`

Description

Use the RunSync method to specify if the analytic instance executing the method should be run in a synchronous manner.

This method is used only to define the transaction in the analytic type definition in optimization. PeopleSoft Analytic Calculation Engine programs uses the Load method for defining synchronous or asynchronous operation.

Parameters

None.

Returns

None.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions

Terminate

Syntax

Terminate()

Description

Use the Terminate method to force the termination of an analytic instance loaded in an analytic server.

This method should only be used from PeopleCode running in a Application Engine program to cause an analytic instance loaded in an analytic server to be terminated.

Attempting to terminate an analytic instance that is not loaded or is loaded in a Application Engine process results in a PeopleCode exception.

Although the Terminate method returns instantly, it may take up to a minute before the analytic instance is actually terminated.

Parameters

None.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," Load, page 21 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 24

Unload

Syntax

Unload()

Description

Use the Unload method to unload the analytic instance executing the method from the analytic server. After unloading the analytic instance, the analytic server process is restarted.

Parameters

None.

Returns

None.

Example

```
&ai.UnLoad();
```

See Also

Chapter 2, "Analytic Calculation Engine Classes," Load, page 21

AnalyticInstance Class Properties

In this section, we discuss the AnalyticInstance properties. The properties are discussed in alphabetical order.

AnalyticType

Description

This property returns the name of the analytic type definition for this AnalyticInstance object as a string.

This property is read-only.

ID

Description

This property returns the analytic instance ID of this AnalyticInstance object as an integer.

This property is read-only.

Messages

Description

This property returns a multi-dimensional array of any, containing the messages that occurred.

The first dimension contains the message set number. The second dimension contains the message number. The third dimension contains the number of parameters in that message. The subsequent dimensions contain the values for the parameters. The maximum number of possible dimensions is 8.

After you access this property, only new messages are returned, that is, messages are only returned once.

This property is read-only.

See Also

Chapter 2, "Analytic Calculation Engine Classes," RunSync, page 23

AnalyticModel Class Methods

In this section, we discuss the AnalyticModel methods. The methods are discussed in alphabetical order.

AddMember

Syntax

```
AddMember(DimName,MemberName)
```

Description

Use the AddMember method to add the specified dimension member to the specified dimension.

Parameters

Parameter	***Description***
DimName	Specify the dimension to which you want to add the member.
MemberName	Specify the name of the member that you want to add to the specified dimension.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," GetMembers, page 31 and Chapter 2, "Analytic Calculation Engine Classes," RenameMember, page 35

AttachTree

Syntax

AttachTree*(DimName,TreeName,SetID, UserKeyValue,EffDt,NodeName,OverrideRecord, DetailStartLvl, TreeDiscardLvl)*

Description

Use the AttachTree method to attach a tree to a dimension. Only one tree can be attached to a dimension at a time. You can only attach a tree before the analytic instance is loaded.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension that you want to attach a tree to.
TreeName	Specify the name of the tree that you want to attach to the dimension.
SetId	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter
EffDt	Specify an effective date for this tree. This parameter takes a string value.

Parameter	Description
NodeName	Specify the tree node used as the dimension hierarchy root. This can be different from the tree root, that is, you can pick a subtree.
OverrideRecord	Specify the name of a record to be used for overriding the aggregate. See *Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine*, "Creating Hierarchies," Working with Overrides.
DetailStartLvl	Specify the tree level number (such as 1 for the node hierarchy root specified by the *NodeName* parameter) at which detail nodes start. If you specify a value other than 0, any nodes at this level or greater than this level are considered details node, and any nodes with levels less than the start level are considered aggregate nodes If you specify 0, detail nodes are nodes that don't have parents (a node can be a parent either by having children or by having child ranges).
TreeDiscardLvl	Specify the tree level number (such as 1 for the node hierarchy root specified by the *NodeName* parameter) at which to stop loading the tree. Nodes at this level or at levels greater are discarded. This parameter is only used if it has a value other than 0, and if it has a value greater than *DetailStartLvl*.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," DetachTree, page 29 and Chapter 2, "Analytic Calculation Engine Classes," GetTree, page 33

Chapter 43, "Tree Classes," page 2291

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Creating Hierarchies," Understanding the Relationship of PeopleSoft Trees to Analytic Models

CalculateCube

Syntax

`CalculateCube``(`*`CubeName`*`[, `*`Sync`*`])`

Description

Use the CalculateCube method to calculate the named cube in synchronous mode.

Parameters

Parameter	***Description***
CubeName	Specify the name of the cube that you want to calculate.
Sync	This parameter is optional and is not used. It is only being included for previous releases. This parameter takes a Boolean value. The value of this parameter is ignored. The cube is always calculated in synchronous mode.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," Running Synchronously, page 14 and Chapter 2, "Analytic Calculation Engine Classes," Recalculate, page 34

DetachTree

Syntax

`DetachTree(DimName)`

Description

Use the DetachTree method to detach a tree from the specified dimension.

You can only attach a tree when the analytic instance is not loaded. This method fails if you try to detach a tree while the analytic instance is loaded.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension from which you want to detach a tree.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AttachTree, page 27 and Chapter 2, "Analytic Calculation Engine Classes," GetTree, page 33

Chapter 43, "Tree Classes," page 2291

GetCubeCollection

Syntax

`GetCubeCollection`*`(CubeCollName)`*

Description

Use the GetCubeCollection method to return a reference to a CubeCollection object.

Parameters

Parameter	***Description***
CubeCollName	Specify the name of the cube collection to which you want a reference.

Returns

A CubeCollection object if successful.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CubeCollection Class, page 36

GetCellProperties

Syntax

`GetCellProperties`*`(CubeName,&Node)`*

Description

Use the GetCellProperties method to get information about a cell.

Parameters

Parameter	***Description***
CubeName	Specify the name of the cube that contains the cell you want information about.
&Node	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail.

Returns

An array of string.

The four strings returned in the array are:

1. Cell type. Values are "Aggregate" or "Detail"
2. User function.
3. Dimension for aggregation, the dimension name.
4. Aggregation reason. Values are: "Dimension Level Override", "Member Level Override", "Cube Dimension Level Override", "None."

See Also

Chapter 8, "Array Class," page 257

GetMembers

Syntax

GetMembers*(DimName,DimFilter)*

Description

Use the GetMembers method to return the names of the members in the specified dimension. You can specify a user function to be used as a filter for the dimension using *DimFilter*.

When filtering hierarchy nodes, if the parent member is filtered, all the child nodes are also filtered.

In order to filter the parent member if all of its children are filtered, the user function should be written such that for the parent node, apply the condition on all of its children, using the FORCHILDREN built-in function. If none of the children satisfy the condition, return zero from the user function, which filters the parent node.

Parameters

Parameter	*Description*
DimName	Specify the name of the dimension from which you want to get the members.
DimFilter	Specify a user function that you want to use to filter the dimension.

Returns

A two-dimensional array of any.

The first dimension contains the member names. The second dimension contains the parent of the member. The members are sorted in the hierarchy order.

If the dimension does not have a hierarchy, all members are returned as level one.

For example, if the dimension has the following members:

```
ALLREGIONS
   NORTH AMERICA
      USA
      CANADA
      MEXICO
   EUROPE
      FRANCE
      GERMANY
      ENGLAND
   ASIA
      JAPAN
      CHINA
```

The GetMembers method returns the following:

Member Name	*Level*
ALLREGIONS	Null
NORTHAMERICA	ALLREGIONS
USA	NORTHAMERICA
CANADA	NORTHAMERICA
MEXICO	NORTHAMERICA
EUROPE	ALLREGIONS
FRANCE	EUROPE
ENGLAND	EUROPE

Member Name	***Level***
GERMANY	EUROPE
ASIA	ALLREGIONS
JAPAN	ASIA
CHINA	ASIA

If the total member is present in the metadata, it is returned as level zero.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AddMember, page 26 and Chapter 2, "Analytic Calculation Engine Classes," RenameMember, page 35

GetTree

Syntax

`GetTree``(DimName)`

Description

Use the GetTree method to return an array of string that contains information about the tree that is attached to the specified dimension.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension to which a tree is attached.

Returns

An array of string.

The array items have the following format:

1. TreeName
2. SetID
3. UserKeyValue

4. EffDt
5. BranchName
6. Override record name
7. DetailStartLvl
8. TreeDiscardLvl

See Also

Chapter 2, "Analytic Calculation Engine Classes," AttachTree, page 27 and Chapter 2, "Analytic Calculation Engine Classes," DetachTree, page 29

Chapter 43, "Tree Classes," page 2291

Recalculate

Syntax

`Recalculate`*`(Sync)`*

Description

Use the Recalculate method to recalculates the loaded analytic model. If *Sync* is true, the transaction is synchronous, else it is performed asynchronously.

The Recalculate method always writes any changes to readable, readable and writable, as well as writable records to the database as specified by the analytic type definition.

If you are running in asynchronous mode, this method returns an optional string that contains the operation ID to be used with the CheckAsyncStatus property.

Parameters

Parameter	*Description*
Sync	Specify whether the recalculation should be run synchronously or asynchronously. This parameter takes a Boolean value: true if it should be run synchronously, false otherwise.

Returns

A string.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CalculateCube, page 28 and Chapter 2, "Analytic Calculation Engine Classes," CheckAsyncStatus, page 16

Chapter 3, "Analytic Calculation Engine Metadata Classes," AnalyticModelDefn, page 58

RenameMember

Syntax

`RenameMember``(`*`DimName,OrigMemberName,NewMemberName`*`)`

Description

Use the RenameMember method to rename a member in the specified dimension.

Parameters

Parameter	***Description***
DimName	Specify the dimension that contains the member you want to rename.
OrigMemberName	Specify the name of the member that you want to rename.
NewMemberName	Specify the new name for the member specified by *OrigMemberName.*

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AddMember, page 26 and Chapter 2, "Analytic Calculation Engine Classes," GetMembers, page 31

AnalyticModel Class Property

In this section, we discuss the AnalyticModel class property.

Messages

Description

This property returns a multi-dimensional array of any containing the messages that occurred during execution of the Recalculate method.

The first dimension contains the message set number. The second dimension contains the message number. The third dimension contains the number of parameters in that message. The subsequent dimensions contain the values for the parameters. The maximum number of possible dimensions is 8.

After you access this property, only new messages are returned, that is, messages are only returned once.

This property is read only.

See Also

Chapter 2, "Analytic Calculation Engine Classes," Recalculate, page 34

CubeCollection Class

Instantiate a CubeCollection object using the AnalyticModel class GetCubeCollection method.

In cube collections, the total member behaves just as the root of the hierarchy. If it's present, it's displayed first in the list.

Filtering of members in a dimension in the cube collection is similar to filtering members using the AnalyticModel class GetMembers method. If you filter the parent node, the children of that node are filtered as well.

See Chapter 2, "Analytic Calculation Engine Classes," GetCubeCollection, page 30.

CubeCollection Class Methods

In this section we discuss the CubeCollection class methods. The methods are described in alphabetical order.

CollapseNode

Syntax

`CollapseNode`*`(DimName,&Node)`*

Description

Use the CollapseNode method to collapse a node in the dimension specified by *DimName*.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that contains the node you want to collapse.

&Node	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail. For example:
	`PRODUCTS     Fax Modem`
	`REGION       East`

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," ExpandNode, page 39

DrillIntoNode

Syntax

`DrillIntoNode`*`(DimName,&Node)`*

Description

Use the DrillIntoNode method to display more data for the specified node and dimension. This is very similar to expanding, except that when you expand a node, the top level (node) is still present. When you drill into a node, the top level or levels are no longer displayed.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that contains the node that you want to drill into.

Parameter	Description
&Node	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail. For example: `PRODUCTS   Fax Modem` `REGION     East`

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," DrillOutOfNode, page 38

DrillOutOfNode

Syntax

`DrillOutOfNode`*`(DimName,&Node)`*

Description

Use the DrillOutOfNode method to collapse the data that you just expanded.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that contains the node you want to drill out of.
&Node	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail. For example: `PRODUCTS     Fax Modem` `REGION       East`

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," DrillIntoNode, page 37

ExpandNode

Syntax

ExpandNode(*DimName*,*&Nodes*,*ExpandAll*)

Description

Use the ExpandNode method to expand the nodes in the specified dimension.

Parameters

Parameter	*Description*
DimName	Specify the name of the dimension contains the node that you want to expand.
&Nodes	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail.
ExpandAll	Specify whether to expand the sub-nodes or not. This parameter takes a Boolean value; true to expand the nodes, false to not expand the nodes.

Returns

None.

Example

In the following example, both the Accounts and Dept dimensions have hierarchies.

```
Account             Dept

All Acoounts            All Depts
         | ---- 100                  |  - ---- A
         |                          |
                | ---- 200                 |------   B -| ---D
         |                      |
                |                      | --- E
                | ---- 300
```

When the data is displayed, it will be as follows:


```
All Accounts
          All Depts
     100
          All Depts
     200
          All Depts
     300
          All Depts
```

When you want to expand All Depts under Account 100 , use the following PeopleSoft Analytic Calculation Engine code:

```
BAMCoordinate coord;
coord.SetValue( "Account", "100");
coord.SetValue("Dept" , "All Depts")
ExpandNode("Dept", coord, false)
```

The result of this will have the following:

```
All Accounts
          All Depts
     100
          All Depts
                   A
                   B
     200
          All Depts
     300
          All Depts
```

See Also

Chapter 2, "Analytic Calculation Engine Classes," CollapseNode, page 36

GetData

Syntax

GetData*(&DataRowset,StartRow,EndRow*[, *NetChanges*])

Description

Use the GetData method to populate a rowset with data from the cube collection.

DataRowset is the rowset into which is put the data. If you specify an already instantiated rowset, the given rowset is populated with the data. If you specify a rowset that has not been instantiated, a new rowset is created, with the record field definition of the desired cube collection.

Parameters

Parameter	***Description***
&DataRowset	Specify the rowset into which to put the data. If you specify an already instantiated rowset, the given rowset is populated with the data. If you specify a rowset that has not been instantiated, a new rowset is created, with the record field definition of the desired cube collection.
StartRow	Specify the first row of data to be retrieved, as an integer.
EndRow	Specify the last row of data to be retrieved, as an integer. If you specify a zero for this parameter, all rows are retrieved.
NetChanges	Specify whether all values within the range are returned, or whether just values changed since the last time the Recalculate method was used. This parameter takes a Boolean value: true, return all values, false, return just the changed values. The default value is false.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetData, page 45 and Chapter 2, "Analytic Calculation Engine Classes," Messages, page 52

Chapter 36, "Rowset Class," page 2127

GetDimFilter

Syntax

GetDimFilter*(DimName)*

Description

Use the GetDimFilter method to return the name of the current user function (filter) used with the specified dimension.

Parameters

Parameter	*Description*
DimName	Specify the name of the dimension for which you want the associated user function (filter).

Returns

A string containing the name of the user function. If there is no filter applied to the dimension, this method returns an empty string.

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetDimFilter, page 46

GetDimSort

Syntax

`GetDimSort``(`*`DimName`*`)`

Description

Use the GetDimSort to return the current sort settings on the specified dimension.

Parameters

Parameter	*Description*
DimName	Specify the name of the dimension for which you'd like the sort settings.

Returns

An array of any. The array contains the same structure as the signature for the SetDimSort method.

Value	*Description*
Array index 1	IsAscending as Boolean
Array index 2	Key1 as string

Value	*Description*
Array index 3	IsAscending2 as Boolean
Array index 4	Key2 as string
Array index 5	IsAscending3 as Boolean
Array index 6	Key3 as string

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetDimSort, page 47

GetLayout

Syntax

`GetLayout`*`(&SlicerArray,&RowAxisArray,&ColumnAxisArray)`*

Description

Use the GetLayout method to return the current layout for the three axes, (slice, row, and column) on the specified dimension.

Parameters

Parameter	*Description*
&SlicerArray	Specify an already instantiated array of string containing the list of fields used on the slice axis.
&RowAxisArray	Specify an already instantiated array of string containing the list of fields used on the row axis.
&ColumnAxisArray	Specify an already instantiated array of string containing the list of fields used on the column axis.

Returns

None.

GetRowCount

Syntax

GetRowCount()

Description

Use the GetRowCount method to return the total number of rows in the cube collection.

The number of rows is a function of the hierarchy, that is, what is shown or not shown, and filtered values.

Parameters

None.

Returns

An integer.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimension, page 77

GetSlice

Syntax

GetSlice*(&SliceRecord)*

Description

Use the GetSlice method to return the current slice values of all the dimensions.

Parameters

Parameter	*Description*
&SliceRecord	Specify an already instantiated record object to hold the values of the slicer record.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetSlice, page 48

SetData

Syntax

`SetData``(&DataRowset)`

Description

Use the SetData method to populate the data in the cube collection with the data in the specified rowset.

The system extracts only changed values from the rowset, that is, values marked as updated. .

SetData method also writes the changes to the database for any record marked as writable or readable and writable in the analytic type definition, and mapped to the specified cube collection.

Parameters

Parameter	Description
&DataRowset	Specify the rowset object that contains the data you want to use to populate the cube collection.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," GetData, page 40

Chapter 35, "Row Class," IsChanged, page 2120

Chapter 36, "Rowset Class," page 2127

SetDimensionOrder

Syntax

SetDimensionOrder_(&FieldNames)_

Description

Use the SetDimensionOrder method to specify the order of dimensions for this cube collection.

Parameters

Parameter	***Description***
&FieldNames	Specify an already instantiated array of string containing the field names that are the dimensions in the cube collection, in the order you want the dimensions.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetDimSort, page 116

SetDimFilter

Syntax

SetDimFilter_(DimName,DimFilter)_

Description

Use the SetDimFilter method to set a dimension filter for the specified dimension in the cube collection.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension for which you want to specify a dimension filter.

Parameter	*Description*
DimFilter	Specify the name of the dimension filter (user function) that you want to use the dimension.

Returns

None.

See Also

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Creating Dimensions," Understanding Dimensions

SetDimSort

Syntax

`SetDimSort``(`*`DimName,IsAscending`*` [, `*`Key1`*`, `*`IsAscending2,Key2,IsAscending3,Key3`*`])`

Description

Use the SetDimSort method to specify the sorting order for the cube collection.

Key1, *Key2*and *Key3* are optional parameters. If they are not specified, the sorting works as a name sort, that is, it sorts by the name of members in the given dimension, either in ascending or descending order.

If keys are specified, the keys values refer to the names of the cubes to be used as they key. The method orders the members of the dimension based on the value of the cubes for each member. The Boolean parameter *IsAscending* defines whether sorting should be in ascending or descending order. Up to three sort keys may be specified. *Key1* is the primary sort key. *Key2* is used to sub-sort any members that have the same key value under *Key1*, and so on.

The scope of this method is the same as the scope of the AnalyticModel object created in PeopleCode, that is, if the AnalyticModel object was declared with scope Global, this method would have a global scope as well, if it was declared as Local, it would have Local, and so on.

Specifying a null string for *DimName* or for keys one through three removes the current value as a sort value.

Parameters

Parameter	*Description*
DimName	Specify the dimension name that you would like sorted.

Parameter	*Description*
IsAscending	Specify if the sort order is ascending or descending. This parameter takes a Boolean value: true if the sort is ascending, false if it's descending.
Key1	Specify the name of a key field by which the sort should be ordered. If you do not specify keys, the sort is ordered by the names of the members in the given dimensions. If you do specify keys, the sort is order by the keys.
IsAscending2	Specify if the sort order for *Key1*is ascending or descending. This parameter takes a Boolean value: true if the sort is ascending, false if it's descending.
Key2	Specify the name of a key field to be used to sub-sort the primary key field *Key1*.
IsAscending3	Specify if the sort order for *Key2*is ascending or descending. This parameter takes a Boolean value: true if the sort is ascending, false if it's descending.
Key3	Specify the name of a key field to be used to sub-sort the primary key field *Key2*.

Returns

None.

SetSlice

Syntax

SetSlice*(&SliceRecord)*

Description

Use the SetSlice method to specify a slice record for this cube collection.

Parameters

Parameter	*Description*
&SliceRecord	Specify an already instantiated record as the record to be used for the slice for the cube collection.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," GetSlice, page 44

ShowHierarchy

Syntax

`ShowHierarchy``(`*`DimName,Show,ExpandToLevel`*`)`

Description

Use the ShowHierarchy method to either hide or display the cube collection hierarchy.

If there is a tree attached to the dimension, and *Show* is false, only the detail level members are shown in the Analytic grid. If *Show* is true, all members including nodes are shown.

When the hierarchy is hidden, the GetData method returns only leaf values. When the hierarchy is displayed, GetData returns node values and leaf values.

The *ExpandToLevel* parameter expands the hierarchy to the level specified. Level indexes start from 0 , the root level.

For example, if you specify *ExpandToLevel* as 1, the grid displays the total (root member) and all the detail members in a dimension with no tree attached.

In the following example, if you specify *ExandToLevel* as 2, the hierarchy would be displayed only to the quarter level, and not to the month level.

```
ALL_TIME
     2003
          Q1
               Jan
               Feb
               Mar
          Q2
               Apr
               May
               Jun
          Q3
               . . .
               . . .
          Q4
     2004
          Q1
          Q2
          Q3
          Q4
```

Parameters

Parameter	*Description*
DimName	Specify the name of the dimension for which you want to either display or hide the hierarchy.
Show	Specify whether to display or hide the hierarchy. This parameter takes a Boolean value; true to display the hierarchy, false to hide it.
ExpandToLevel	Specify to what level to expand the hierarchy, as a number.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CollapseNode, page 36; Chapter 2, "Analytic Calculation Engine Classes," ExpandNode, page 39 and Chapter 2, "Analytic Calculation Engine Classes," GetData, page 40

UnsetDimFilter

Syntax

`UnsetDimFilter``(`*`DimName`*`)`

Description

Use the UnsetDimFilter method to remove the filter on the specified dimension for the cube collection.

Parameters

Parameter	*Description*
DimName	Specify the name of the dimension from which you want to remove the filter.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetDimFilter, page 46

UnsetDimSort

Syntax

`UnsetDimSort`*`(DimName)`*

Description

Use the UnsetDimSort method to remove the sorting on the specified dimension for a cube collection. After using this method, the members are returned in the global member order as specified in the analytic model definition.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension from which you want to remove sorting.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetDimSort, page 47

CubeCollection Class Property

In this section we discuss the CubeCollection class property.

Messages

Description

This property returns a multi-dimensional array of any containing the messages that occurred during execution of the GetData, SetData, and CalculateCube methods.

The first dimension contains the message set number. The second dimension contains the message number. The third dimension contains the number of parameters in that message. The subsequent dimensions contain the values for the parameters. The maximum number of possible dimensions is 8.

After you access this property, only new messages are returned, that is, messages are only returned once.

This property is read-only.

See Also

Chapter 2, "Analytic Calculation Engine Classes," GetData, page 40 and Chapter 2, "Analytic Calculation Engine Classes," SetData, page 45

Chapter 2, "Analytic Calculation Engine Classes," CalculateCube, page 28

Chapter 3

Analytic Calculation Engine Metadata Classes

This chapter provides an overview of the Analytic Calculation Engine Metadata classes, and discuses:

- Using the Analytic Calculation Engine Metadata classes
- Error handling
- Data types of the Analytic Calculation Engine Metadata objects
- Scope of Analytic Calculation Engine Metadata objects
- How to import the Analytic Calculation Engine Metadata classes
- How to create an Analytic Calculation Engine Metadata class object
- Analytic Calculation Engine Metadata classes reference

Important! The Analytic Calculation Engine Metadata classes are not supported on IBM z/OS and Linux for IBM System z platforms.

Understanding the PeopleSoft Analytic Calculation Engine Metadata Classes

PeopleSoft Analytic Calculation Engine comprises a calculation engine plus several PeopleTools features which enable application developers to define both the calculation rules and the display of calculated data within PeopleSoft applications for the purposes of multi-dimensional reporting, data editing and analysis.

More specifically, developers create *analytic models* in order to define the rules which are used to calculate data. Developers also create PeopleSoft Pure Internet Architecture pages with *analytic grids* in order to display the data within PeopleSoft applications. End users view, analyze and make changes to this data. When end users save their changes, PeopleSoft Analytic Calculation Engine recalculates the data and saves the calculated data to the database.

PeopleCode enables developers to manipulate analytic calculation data as follows:

- Use the Analytic Calculation Engine classes to either retrieve or specify data in an instance of an analytic model loaded into the system, and also to calculate (or recalculate) cube values.
- Use the Analytic Calculation Engine metadata classes to manipulate an analytic model definition. For example, you can add cubes to a cube collection or rename an existing user function for a model.

- Use the Analytic Grid classes to manipulate the display of analytic calculation data on a page.
- Use the Analytic Type classes to manipulate an analytic type definition. For example, you can specify a new analytic model for an analytic type definition.

See Also

Chapter 2, "Analytic Calculation Engine Classes," page 13

Chapter 4, "Analytic Grid Classes," page 159

Chapter 5, "Analytic Type Classes," page 169

Using the Analytic Calculation Engine Metadata Classes

You can create analytic model definitions using Application Designer. PeopleSoft provides the Analytic Calculation Engine Metadata classes for accessing these definitions at runtime.

You can also create an analytic model definition in PeopleCode, using the Create method, then saving it to the database using the Save method. After you save the definition, you can access it using Application Designer.

All of the primary objects (dimensions, cube collections, cubes, and so on) are instantiated from an analytic model object, so you only need to create an instance of the analytic model. You can also build a rule as an object instead of as a string.

Inserting and Deleting Objects

If you insert or delete an object from a group of objects, any existing object references are no longer valid. For example, if you use the DeleteCube method, any existing references to cube objects are no longer valid. You need to create any object references after using any insert or delete method.

Building a Rule as an Object

The RuleExpressions sub-package contains several classes. Each of these classes represents the different parts of the analytic calculation engine rule grammar. These objects are created using the Create built-in function and the constructor for that class, then added to the RuleDefn object with the AddRuleExpression method.

The following example creates a cube that uses the Analytic Calculation Engine built-in function AT.


```
Function AT_Cube
   /** AT Cube Rule **/
   &Expected = "AT(GENERAL2, [GENERAL2:4], LOOKUPVALUES)";
   &CubeName = "AT";

   &FunCall = create PT_ANALYTICMODELDEFN:RuleExpressions:FunctionCall⇒
(&Constants.FuncCall_Type_Builtin, &Constants.FuncCall_Builtin_AT);

   &Constant = create PT_ANALYTICMODELDEFN:RuleExpressions:Constant⇒
(&Constants.Constant_Type_Literal, "GENERAL2");
   &FunCall.AddArgument(&Constant);

   &MemberRefArgument = create PT_ANALYTICMODELDEFN:RuleExpressions:⇒
MemberReference("GENERAL2", "4");
   &FunCall.AddArgument(&MemberRefArgument);

   &CubeArgument = create PT_ANALYTICMODELDEFN:RuleExpressions:Cube("LOOKUPVALUES");
   &FunCall.AddArgument(&CubeArgument);

   &RuleDefn = create PT_ANALYTICMODELDEFN:RuleDefn("");
   &CubeDefn = &Model.GetCube(&CubeName);
   &RuleDefn.AddRuleExpression(&FunCall);
   &RuleDefn.GenerateRule();
   &CubeDefn.SetRule(&RuleDefn);
   If ( Not (Exact(&RuleDefn.RuleString, &Expected))) Then
      &Model.Save();
      Error ("Rule Experession API Failed [Cube " | &CubeName | " Rule]");
   End-If;
End-Function;
```

Error Handling

All the Analytic Calculation Engine Metadata classes throw PeopleCode exceptions for any fatal error that occurs in the execution of the operation. PeopleSoft recommends enclosing your analytic model programs in try-catch statements. This way, if your program catches the exception, the message set and message number that are associated with the exception object indicate the error.

See Also

Chapter 16, "Exception Class," Try-Catch Sections, page 764

Data Types of the Analytic Calculation Engine Metadata Objects

Every PeopleSoft Analytic Calculation Engine metadata object is declared as its own data type, that is, AnalyticModelDefn objects are declared as type AnalyticModelDefn, CubeDefn objects are declared as type CubeDefn, and so on.

Scope of Analytic Calculation Engine Metadata Objects

The Analytic Calculation Engine Metadata objects can only be instantiated from PeopleCode.

These objects can be used anywhere you have PeopleCode, that is, in a Application Engine program, an application class, record field PeopleCode, and so on.

Analytic Calculation Engine Metadata objects can be of scope Local, Component or Global.

How to Import the Analytic Calculation Engine Metadata Classes

The Analytic Calculation Engine metadata classes are *not* built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing the Analytic Calculation Engine metadata classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_ANALYTICMODELDEFN contains the following subclasses:

- AnalyticModelDefn
- CubeCollectionDefn
- CubeDefn
- DimensionDefn
- ExplicitDimSet
- OrganizerDefn
- RuleDefn
- UserFunctionDefn

Additionally, there is a sub-application package, RuleExpressions, that contains classes used to build a rule as object instead of as a simple string. This package contains the following classes:

- Assignment
- Comparison
- Constant
- Constants
- Cube
- ExpressionBlock

- FunctionCall
- MemberReference
- Operation
- RuleExpression
- Variable

The import statements that you should use should be as follows:

```
import PT_ANALYTICMODELDEFN:*;
import PT_ANALYTICMODELDEFN:RuleExpressions:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available.

See Also

Chapter 6, "Application Classes," page 189

How to Create an Analytic Calculation Engine Metadata Class Object

After you've imported the Analytic Calculation Engine metadata classes, you need to instantiate an instance of the AnalyticModelDefn class, using the constructor for that class and the Create function. After you create the object, you must populate it using either the Get or Create method.

The following example creates an AnalyticModelDefn object from the QE_ALLFUNCTION analytic model definition.

```
import PT_ANALYTICMODELDEFN:*;

Local AnalyticModelDefn &Model;

&Model = create AnalyticModelDefn("QE_ALLFUNCTION");
```

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Analytic Calculation Engine Metadata Classes Constructor, page 57

Analytic Calculation Engine Metadata Classes Constructor

You must use the constructor for the AnalyticModelDefn class to instantiate an instance of that class. All other metadata objects are instantiated from this class.

AnalyticModelDefn

Syntax

AnalyticModelDefn*(ModelName)*

Description

Use the AnalyticModelDefn constructor to create an AnalyticModelDefn object.

Once the AnalyticModelDefn object is created, you can then execute either the Get or Create methods to 'instantiate' the model.

Note. The Delete and Rename methods can only be executed before a model is 'instantiated'.

Parameters

Parameter	***Description***
ModelName	Specify the name of an analytic model definition.

Returns

An AnalyticModelDefn object.

Example

```
&Model = create AnalyticModelDefn("QE_ALLFUNCTION");
```

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AnalyticModelDefn Class, page 58

AnalyticModelDefn Class

Use the AnalyticModelDefn class to view or manipulate an analytic model definition that's already been created in Application Designer, or to create a new definition.

See Also

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine

AnalyticModelDefn Class Methods

The following section discusses the AnalyticModelDefn class methods. The methods are discussed in alphabetical order.

AddCube

Syntax

`AddCube`*`(CubeName)`*

Description

Use the AddCube method to add a new cube to the analytic model definition. This method fails if the cube specified by *CubeName* already exists.

Parameters

Parameter	***Description***
CubeName	Specify the name of a new cube that you want to add to the analytic model.

Returns

A CubeDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCube, page 63; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCube, page 69; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 74; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 77 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 84

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 98

AddCubeCollection

Syntax

`AddCubeCollection`*`(CubeCollectionName)`*

Description

Use the AddCubeCollection method to add a new cube collection to the analytic model definition. This method fails if *CubeCollectionName* already exists.

Parameters

Parameter	*Description*
CubeCollectionName	Specify the name of a new cube collection that you want to add to the analytic model definition.

Returns

A CubeCollectionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCubeCollection, page 64; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCubeCollection, page 70; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 75 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCubeCollection, page 85

AddDimension

Syntax

`AddDimension``(`*`DimName`*`)`

Description

Use the AddDimension method to add a new dimension to the analytic model. This method fails if the specified dimension already exists.

When you add a dimension to a cube, the dimension is also automatically added to any cube collections that already contain the cube.

Parameters

Parameter	*Description*
DimName	Specify the name of the dimension you want to add. This must be a new dimension.

Returns

A DimensionDef object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 65; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 71 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimension, page 77

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 93

AddExplicitDimensionSet

Syntax

`AddExplicitDimensionSet`*`(ExplicitDimSetName)`*

Description

Use the AddExplicitDimensionSet method to add a new explicit dimension set to the analytic model. This method fails if the specified explicit dimension set already exists.

Parameters

Parameter	***Description***
ExplicitDimSetName	Specify the name of the explicit dimension you want to add. This must be a new explicit dimension.

Returns

An ExplicitDimensionSet object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSet, page 79; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteExplicitDimensionSet, page 71; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyExplicitDimensionSet, page 65; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSetNames, page 80 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameExplicitDimensionSet, page 86

AddOrganizer

Syntax

`AddOrganizer``(OrganizerName)`

Description

Use the AddOrganizer method to add a new organizer to the analytic model. This method fails if the organizer specified by *OrganizerName* already exists.

Parameters

Parameter	***Description***
OrganizerName	Specify the name of the organizer you want to add.

Returns

An OrganizerDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteOrganizer, page 72; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizer, page 80; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizerNames, page 81 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameOrganizer, page 87

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 125

AddUserFunction

Syntax

`AddUserFunction``(UserFunctionName)`

Description

Use the AddUserFunction method to add a new user function to the analytic model. This method fails if the user function specified by *UserFunction* already exists.

Parameters

Parameter	*Description*
UserFunctionName	Specify the name of the new user function that you want to add.

Returns

A UserFunction object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 67; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 73; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 82; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 82 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 88

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 123

CopyCube

Syntax

CopyCube*(CubeName,NewCubeName)*

Description

Use the CopyCube method to copy the specified cube to the new cube. This method fails if the cube specified by *CubeName* already exists.

Parameters

Parameter	*Description*
CubeName	Specify the name of the cube that you want to copy.
NewCubeName	Specify the name of the new cube that you want to create.

Returns

A CubeDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 74; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 77 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 84

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 98

CopyCubeCollection

Syntax

`CopyCubeCollection``(`*`CubeCollectionName,NewCubeCollectionName`*`)`

Description

Use the CopyCubeCollection method to copy the specified cube collection to a new collection. If the new cube collection specified by *NewCubeCollectionName* already exists, this method fails.

Parameters

Parameter	***Description***
CubeCollectionName	Specify the name of the cube collection that you want to copy.
NewCubeCollectionName	Specify the name you want for the new cube collection.

Returns

A CubeCollectionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCubeCollection, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCubeCollection, page 70; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 75; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 76 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCubeCollection, page 85

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 108

CopyDimension

Syntax

CopyDimension(*DimName*,*NewDimName*)

Description

Use the CopyDimension method to copy the dimension specified by *DimName* to a new dimension. If the dimension specified by *NewDimName* already exists, this method fails.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension that you want to copy.
NewDimName	Specify the name of the new dimension that you want the data copied to.

Returns

A DimensionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 60; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 71 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameDimension, page 85

CopyExplicitDimensionSet

Syntax

CopyExplicitDimensionSet(*ExplicitDimSetName*,*NewExplicitDimSetName*)

Description

Use the CopyExplicitDimensionSet method to copy the explicit dimension set specified by *ExplicitDimSetName* to a new explicit dimension set. If the explicit dimension set specified by *NewExplicitDimSetName* already exists, this method fails.

Parameters

Parameter	*Description*
ExplicitDimSetName	Specify the name of the explicit dimension set that you want to copy.
NewExplicitDimSetName	Specify the name of the new explicit dimension set that you want the data copied to.

Returns

An ExplicitDimensionSet object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameExplicitDimensionSet, page 86; Chapter 3, "Analytic Calculation Engine Metadata Classes," AddExplicitDimensionSet, page 61; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteExplicitDimensionSet, page 71; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSet, page 79 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSetNames, page 80

CopyTo

Syntax

`CopyTo``(`*`NewModelName`*`)`

Description

Use the CopyTo method to copy the AnalyticModelDefn object to a new analytic model specified by *NewModelName.* If the model set specified by *NewModelName* already exists, this method fails.

Parameters

Parameter	*Description*
NewModelName	Specify the name of a new analytic model definition that you want to create.

Returns

An AnalyticModelDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 68; Chapter 3, "Analytic Calculation Engine Metadata Classes," Delete, page 68; Chapter 3, "Analytic Calculation Engine Metadata Classes," Get, page 74; Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 83 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 89

CopyUserFunction

Syntax

`CopyUserFunction``(`*`UserFunctionName,NewUserFunctionName`*`)`

Description

Use the CopyUserFunction method to copy the user function specified by *UserFunctionName* to a new user function. If the user function specified by *NewUserFunctionName* already exists, this method fails.

Parameters

Parameter	***Description***
UserFunctionName	Specify the name of the user function that you want to copy.
NewUserFunctionName	Specify the name of the new user function that you want to add.

Returns

A UserFunctionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 62; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 73; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 82; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 82 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 88

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 123

Create

Syntax

`Create()`

Description

Use the Create method to create, and instantiate, a new model. If you save the new model after you create it, you can then access it in Application Designer as an analytic model definition.

If the model already exists, an exception is thrown.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyTo, page 66; Chapter 3, "Analytic Calculation Engine Metadata Classes," Delete, page 68; Chapter 3, "Analytic Calculation Engine Metadata Classes," Get, page 74 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 83

Delete

Syntax

`Delete()`

Description

Use the Delete method to delete the analytic model executing the method.

You can only use this method on a *closed* analytic model definition, that is, before you use a Get or Create method.

Warning! The delete occurs immediately, that is, the analytic model definition is removed from the database. Only use this method if you're certain you want to delete the definition.

If this model is used by any existing analytic type definition, this method fails.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyTo, page 66; Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 68; Chapter 3, "Analytic Calculation Engine Metadata Classes," Get, page 74; Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 83 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 89

DeleteCube

Syntax

`DeleteCube``(CubeName,ForceDelete)`

Description

Use the DeleteCube method to delete the cube specified by *CubeName.*

Parameters

Parameter	***Description***
CubeName	Specify the name of the cube you want to delete from the analytic model.
ForceDelete	Specify if the cube should always be deleted. This parameter takes a Boolean value. If you specify *ForceDelete* as false, and the cube is used by another part (such as a cube collection) this method fails. If you specify *ForceDelete* as true, and the cube is used by another part, the cube is still deleted.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 74; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 77 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 84

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 108

DeleteCubeCollection

Syntax

`DeleteCubeCollection``(`*`CubeCollectionName,ForceDelete`*`)`

Description

Use the DeleteCubeCollection method to delete the cube collection specified by *CubeCollectionName.*

Parameters

Parameter	*Description*
CubeCollectionName	Specify the name of the cube collection that you want to delete from the analytic model.
ForceDelete	Specify if the cube collection should always be deleted. This parameter takes a Boolean value. If you specify *ForceDelete* as false, and the cube collection is used by another part (such as an Organizer), this method fails. If you specify *ForceDelete* as true, and the cube collection is used by another part, the cube collection is still deleted.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCubeCollection, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 75; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 76 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCubeCollection, page 85

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 108

DeleteDimension

Syntax

DeleteDimension(*DimensionName*,*ForceDelete*)

Description

Use the DeleteDimension method to delete the dimension specified by *DimensionName.*

Parameters

Parameter	***Description***
DimensionName	Specify the name of the dimension you want to delete.
ForceDelete	Specify if the dimension should always be deleted. This parameter takes a Boolean value. If you specify *ForceDelete* as false, and the dimension is used by another part (such as a cube), this method fails. If you specify *ForceDelete* as true, and the dimension is used by another part, the dimension is still deleted.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 60; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 65; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimension, page 77; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 78 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameDimension, page 85

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 93

DeleteExplicitDimensionSet

Syntax

DeleteExplicitDimensionSet(*ExplicitDimensionSetName*,*ForceDelete*)

Description

Use the DeleteExplicitDimensionSet method to delete the explicit dimension set specified by *ExplicitDimensionSetName.*

Parameters

Parameter	***Description***
ExplicitDimensionSetName	Specify the name of the explicit dimension set you want to delete.
ForceDelete	The value of this parameter is ignored in the current release.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddExplicitDimensionSet, page 61; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyExplicitDimensionSet, page 65; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSet, page 79; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSetNames, page 80 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameExplicitDimensionSet, page 86

DeleteOrganizer

Syntax

`DeleteOrganizer``(`*`OrganizerName`*`,`*`ForceDelete`*`)`

Description

Use the DeleteOrganizer method to delete the organizer specified by *OrganizerName.*

Parameters

Parameter	***Description***
OrganizerName	Specify the name of the organizer you want to delete.
ForceDelete	This value is ignored in the current release.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddOrganizer, page 62; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizer, page 80; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizerNames, page 81 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameOrganizer, page 87

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 125

DeleteUserFunction

Syntax

`DeleteUserFunction``(`*`UserFunctionName,ForceDelete`*`)`

Description

Use the DeleteUserFunction method to delete the user function specified by *UserFunctionName.*

Parameters

Parameter	***Description***
UserFunctionName	Specify the name of the user function you want to delete.
ForceDelete	Specify if the user function should always be deleted. This parameter takes a Boolean value. If you specify *ForceDelete* as false, and the user function is used by another part, this method fails. If you specify *ForceDelete* as true, and the user function is used by another part, the user function is still deleted.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 62; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 67; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 82; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 82 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 88

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 123

Get

Syntax

Get()

Description

Use the Get method to instantiate an instance of the analytic model.

If the model doesn't exist, an exception is thrown.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyTo, page 66; Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 68; Chapter 3, "Analytic Calculation Engine Metadata Classes," Delete, page 68; Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 83 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 89

GetCube

Syntax

GetCube*(CubeName)*

Description

Use the GetCube method to return a reference to the cube specified by *CubeName*. This method fails if the cube specified by *CubeName* doesn't exist.

Parameters

Parameter	*Description*
CubeName	Specify the name of the cube that you want a reference to.

Returns

A CubeDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCube, page 63; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCube, page 69; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 77 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 84

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 108

GetCubeCollection

Syntax

```
GetCubeCollection(CubeCollectionName)
```

Description

Use the GetCubeCollection method to return a reference to a cube collection. This method fails if the cube collection specified by *CubeCollectionName* doesn't exist.

Parameters

Parameter	*Description*
CubeCollection	Specify the name of the cube collection you want a reference to.

Returns

A CubeCollectionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCubeCollection, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCubeCollection, page 64; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCubeCollection, page 70; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 75; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 76 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCubeCollection, page 85

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 108

GetCubeCollectionNames

Syntax

GetCubeCollectionNames()

Description

Use the GetCubeCollectionNames method to return an array of strings containing all the names of the cube collections associated with this analytic model.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCubeCollection, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCubeCollection, page 64; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCubeCollection, page 70; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 75 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCubeCollection, page 85

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 108

Chapter 8, "Array Class," page 257

GetCubeNames

Syntax

`GetCubeNames``()`

Description

Use the GetCubeNames method to return an array of strings containing all the names of the cubes associated with this analytic model.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCube, page 63; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCube, page 69; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 74 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 84

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 108

Chapter 8, "Array Class," page 257

GetDimension

Syntax

`GetDimension`*`(DimName)`*

Description

Use the GetDimension method to return a reference to the dimension specified by *DimName*. This method fails if the dimension specified by *DimName* doesn't exist.

Parameters

Parameter	*Description*
DimName	Specify the name of the dimension that you want a reference to.

Returns

A DimensionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 60; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 65; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 71; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 78 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameDimension, page 85

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 93

GetDimensionNames

Syntax

`GetDimensionNames``()`

Description

Use the GetDimensionNames method to return an array of string containing the names of all the dimensions associated with the analytic model.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 60; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 65; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 71; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 71; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimension, page 77 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameDimension, page 85

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 93

Chapter 8, "Array Class," page 257

GetExplicitDimensionSet

Syntax

`GetExplicitDimensionSet`*`(ExplicitDimSetName)`*

Description

Use the GetExplicitDimensionSet method to return a reference to the explicit dimension set specified by *ExplicitDimSetName.* This method fails if the explicit dimension set specified by *ExplicitDimSetName* doesn't exist.

Parameters

Parameter	***Description***
ExplicitDimSetName	Specify the name of the explicit dimension set that you want a reference to.

Returns

An ExplicitDimensionSet object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddExplicitDimensionSet, page 61; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyExplicitDimensionSet, page 65; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteExplicitDimensionSet, page 71 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSetNames, page 80

GetExplicitDimensionSetNames

Syntax

`GetExplicitDimensionSetNames``()`

Description

Use the GetExplicitDimensionSetNames method to return an array of string containing the names of all the explicit dimension sets associated with the analytic model.

Parameters

None.

Returns

An array of strings.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," ExplicitDimensionSet Class, page 95

GetOrganizer

Syntax

`GetOrganizer``(`*`OrganizerName`*`)`

Description

Use the GetOrganizer method to return a reference to the organizer specified by *OrganizerName.* This method fails if the organizer specified by *OrganizerName* doesn't exist.

Parameters

Parameter	***Description***
OrganizerName	Specify the name of the organizer you want a reference to.

Returns

An OrganizerDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddOrganizer, page 62; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteOrganizer, page 72; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizerNames, page 81 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameOrganizer, page 87

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 125

GetOrganizerNames

Syntax

```
GetOrganizerNames()
```

Description

Use the GetOrganizerNames method to return an array of string containing the names of all the organizers associated with the analytic model.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddOrganizer, page 62; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteOrganizer, page 72; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizer, page 80 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameOrganizer, page 87

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 125

Chapter 8, "Array Class," page 257

GetUserFunction

Syntax

`GetUserFunction``(`*`UserFunctionName`*`)`

Description

Use the GetUserFunction method to return a reference to the user function specified by *UserFunctionName.* This method fails if the user function specified by *UserFunctionName* doesn't exist.

Parameters

Parameter	***Description***
UserFunctionName	Specify the name of a user function that you want a reference to.

Returns

A UserFunctionDefn object

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 62; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 67; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 73; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 82 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 88

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 123

GetUserFunctionNames

Syntax

`GetUserFunctionNames``()`

Description

Use the GetUserFunctionNames method to return an array of string containing the names of all the user functions associated with the analytic model.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 62; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 67; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 73; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 82 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 88

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 123

Chapter 8, "Array Class," page 257

Rename

Syntax

```
Rename(NewModelName)
```

Description

Use the Rename method to rename the analytic model.

You can only use this method on a *closed* analytic model definition, that is, before you use a Get or Create method.

Parameters

Parameter	*Description*
NewModelName	Specify the new name of the model.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 68; Chapter 3, "Analytic Calculation Engine Metadata Classes," Get, page 74 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 89

RenameCube

Syntax

RenameCube(*CubeName*,*NewCubeName*,*ForceRename*)

Description

Use the RenameCube method to rename the cube specified by *CubeName* to *NewCubeName.*

Parameters

Parameter	***Description***
CubeName	Specify the name of the cube you want to rename.
NewCubeName	Specify the new name for the cube.
ForceRename	Specify if the cube should always be renamed. This parameter takes a Boolean value. If you specify *ForceRename* as false, and the cube is used by another part (such as a cube collection) this method fails. If you specify *ForceRename* as true, and the cube is used by another part, the cube is still renamed.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCube, page 63; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCube, page 69; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 74 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 77

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 98

RenameCubeCollection

Syntax

RenameCubeCollection*(CubeCollectionName,NewCubeCollectionName, ForceRename*

Description

Use the RenameCubeCollection method to rename the cube collection specified by *CubeCollectionName* to the new name *NewCubeCollectionName*

Parameters

Parameter	***Description***
CubeCollectionName	Specify the name of the cube collection that you want to rename.
NewCubeCollectionName	Specify the new name of the cube collection.
ForceRename	Specify if the cube collection should always be renamed. This parameter takes a Boolean value. If you specify *ForceRename* as false, and the cube collection is used by another part (such as an organizer) this method fails. If you specify *ForceRename* as true, and the cube collection is used by another part, the cube collection is still renamed.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCubeCollection, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCubeCollection, page 64; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCubeCollection, page 70; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 75 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 76

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 76

RenameDimension

Syntax

RenameDimension*(DimName,NewDimName,ForceRename)*

Description

Use the RenameDimension method to rename the dimension specified by *DimName* to the new name *NewDimName*.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension you want to rename.
NewDimName	Specify the new name of the dimension.
ForceRename	Specify if the dimension should always be renamed. This parameter takes a Boolean value. If you specify *ForceRename* as false, and the dimension is used by another part (such as a cube) this method fails. If you specify *ForceRename* as true, and the dimension is used by another part, the dimension is still renamed.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 60; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 65; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 71; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimension, page 77 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 78

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 93

RenameExplicitDimensionSet

Syntax

```
RenameExplicitDimensionSet(ExplicitDimSetName,NewExplicitDimSetName,
ForceRename)
```

Description

Use the RenameExplicitDimensionSet method to rename the explicit dimension set specified by *ExplicitDimSetName* to the new name *NewExplicitDimSetName*.

Parameters

Parameter	*Description*
ExplicitDimSetName	Specify the name of the explicit dimension set you want to rename.
NewExplicitDimSetName	Specify the new name of the explicit dimension set.
ForceRename	The value for this parameter is ignored in the current release.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddExplicitDimensionSet, page 61; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyExplicitDimensionSet, page 65; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteExplicitDimensionSet, page 71; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSet, page 79 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameExplicitDimensionSet, page 86

RenameOrganizer

Syntax

`RenameOrganizer``(OrganizerName,NewOrganizerName, ForceRename)`

Description

Use the RenameOrganizer method to rename the organizer specified by *OrganizerName* to the new name *NewOrganizerName.*

Parameters

Parameter	*Description*
OrganizerName	Specify the name of the organizer you want to rename.
NewOrganizerName	Specify the new name for the organizer.
ForceRename	The value for this parameter is ignored in the current release.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddOrganizer, page 62; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteOrganizer, page 72; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizer, page 80 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizerNames, page 81

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 125

RenameUserFunction

Syntax

`RenameUserFunction``(UserFunctionName,NewUserFunctionName, ForceRename)`

Description

Use the RenameUserFunction method to rename the user function specified by *UserFunctionName* to the new name *NewUserFunctionName.*

Parameters

Parameter	***Description***
UserFunctionName	Specify the name of the user function you want to rename.
NewUserFunctionName	Specify the new name for the user function.
ForceRename	Specify if the user function should always be renamed. This parameter takes a Boolean value. If you specify *ForceRename* as false, and the user function is used by another part this method fails. If you specify *ForceRename* as true, and the user function is used by another part, the user function is still renamed.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 62; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 67; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 73; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 82 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 82

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 123

Save

Syntax

Save()

Description

Use the Save method to save any changes to the analytic model definition to the database. If the saved model is valid, the value of the IsValid property is set to true, if it is not valid, it is set to false.

The Messages property is populate with messages from the validation that occurs with the save.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 68 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Validate, page 89

Chapter 3, "Analytic Calculation Engine Metadata Classes," IsValid, page 91 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Messages, page 92

Validate

Syntax

Validate()

Description

Use the Validate method to validate the analytic model definition. This method returns true if the model successfully validates. If this method returns false, the detailed error messages are available through the Messages property.

The IsValid property is not set with this method. The Save method sets the IsValid property.

Parameters

None.

Returns

A Boolean value: true if the analytic model definition validates successfully, false otherwise.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 68; Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 89; Chapter 3, "Analytic Calculation Engine Metadata Classes," Messages, page 92 and Chapter 3, "Analytic Calculation Engine Metadata Classes," IsValid, page 91

AnalyticModelDefn Properties

The following section discusses the AnalyticModelDefn class properties. The properties are discussed in alphabetical order.

CircularFormulaWarn

Description

Use this property to specify whether warnings occur when there are circular dependencies as rules are added. This property takes a Boolean value: true if warnings should occur, false otherwise. The default value for this property is false.

This property is read-write.

Description

Description

Use this property to specify the description of the analytic model. This property takes a string value.

This property is read-write.

IsValid

Description

This property indicates if the analytic model is valid, that is, if it would successfully load.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Validate, page 89 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 89

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Creating Analytic Model Definitions," Validating Analytic Models

LongDescription

Description

Use this property to specify the long description of the analytic model definition. This property takes a string value.

This property is read-write.

MaxDelta

Description

This property specifies the maximum number of value changes for the analytic model

This property takes a float value. The default value is 0.

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Creating Analytic Model Definitions"

MaxIterations

Description

Use this property to specify the maximum number of iterations for the analytic model.

This property take an integer value. The default value is 100.

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Designing and Editing Analytic Models"

Messages

Description

This property returns a multi-dimensional array of any that contains the messages that occurred during execution of the Validate or Save methods. This array is only populated after the Validate or Save method has completed successfully.

The first dimension contains the message set number. The second dimension contains the message number. The third dimension contains the number of parameters in that message. The subsequent dimensions contain the values for the parameters. The maximum number of possible dimensions is 8.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Validate, page 89 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 89

Name

Description

This property indicates the name of the analytic model.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 83

ResolveCircularDeps

Description

Use this property to specify if the analytic model should resolve circular dependencies through iteration. This property takes a Boolean value: true if the dependencies should be resolved, false otherwise. The default value is false.

This property is read-write.

DimensionDefn Class

Use the DimensionDefn class to access a dimension that's associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn methods:

- AddDimension
- CopyDimension
- GetDimension

DimensionDefn Class Properties

The following section discusses the DimensionDefn class properties. The properties are discussed in alphabetical order.

AggregateSequence

Description

This property indicates the sequence number of this dimension used during aggregation of the analytic model

This property is read-only..

AggregationUserFunction

Description

Use this property to specify the name of a user function that can be used to override the default aggregation for this dimension.

This property is read-write.

Comments

Description

Use this property to specify additional notes about the dimension.

This property is read-write.

Name

Description

This property indicates the name of the dimension.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 78 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameDimension, page 85

TotalMemberName

Description

Use this property to specify a field value that is used at runtime to host the total value for this dimension. If a total isn't applicable, this property has no value.

This property is read-write.

ExplicitDimensionSet Class

Use the ExplicitDimensionSet class to access an explicit dimension set associated with an analytic model. You instantiate an object of this class using one of the following AnalyticModelDefn methods:

- AddExplicitDimensionSet
- CopyExplicitDimensionSet
- GetExplicitDimensionSet

You can also use the GetExplicitDimensionSetNames to return an array containing the names of all the explicit dimension sets associated with the analytic model.

ExplicitDimensionSet Methods

The following section discusses the ExplicitDimensionSet class methods. The methods are discussed in alphabetical order.

AttachDimension

Syntax

`AttachDimension`*`(DimName)`*

Description

Use the AttachDimension method to attach a dimension to an explicit dimension set.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension you want to attach.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 96

DetachDimension

Syntax

`DetachDimension`*`(DimName)`*

Description

Use the DetachDimension method to detach a dimension from an explicit dimension set.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension that you want to detach.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 95

GetDimensionNames

Syntax

`GetDimensionNames``()`

Description

Use the GetDimensionNames method to return an array of string that contains the names of all the dimensions associated with the ExplicitDimensionSet.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 95 and Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 96

ExplicitDimensionSet Properties

The following section discusses the ExplicitDimensionSet class properties. The properties are discussed in alphabetical order.

Name

Description

The Name property returns the name of the explicit dimension set as a string.

This property is read-only.

SequenceNumber

Description

This property returns the sequence number of the explicit dimension set.

This property is read-only.

CubeDefn Class

Use the CubeDefn class to access a cube associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn methods:

- AddCubeDefn
- CopyCubeDefn
- GetCubeDefn

CubeDefn Class Methods

The following section discusses the CubeDefn class methods. The methods are discussed in alphabetical order.

AttachDimension

Syntax

`AttachDimension`*`(DimName)`*

Description

Use the AttachDimension method to attach an existing dimension specified by *DimName* to the cube.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension that you want to add.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 99; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 103 and Chapter 3, "Analytic Calculation Engine Metadata Classes," UsesDimension, page 105

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 60

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 93

DetachDimension

Syntax

`DetatchDimension`*`(DimName)`*

Description

Use the DetachDimension method to detach the dimension specified by *DimName* from the cube.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension that you want to detach from the cube.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 98; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 103 and Chapter 3, "Analytic Calculation Engine Metadata Classes," UsesDimension, page 105

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 60

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 93

GetCauses

Syntax

`GetCauses`*`(CauseType)`*

Description

Any cube that affects another cube is a *cause* of that cube. Use the GetCauses method to return a list of all the other cubes that are the causes for this cube.

The cube names are returned in an array of string.

Parameters

Parameter	***Description***
CauseType	Specify the type of causes you want returned.

Value	***Description***
AnalyticModel_DirectCauses	Only return direct causes.
AnalyticModel_AllCauses	Return all cubes that are causes for this cube, both direct and indirect.
AnalyticModel_AllInputs	Return all cubes that have input to this cube.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetEffects, page 102 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCircularDeps, page 101

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Creating Data Cubes," Understanding Causes and Inputs

GetCircularDeps

Syntax

`GetCircularDeps`*`(DimName)`*

Description

Use the GetCircularDeps method to return all cubes that have circular dependencies based on the dimension specified by *DimName.*

The cube names are returned in an array of string.

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension that you want to find circular dependencies for.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCauses, page 100 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetEffects, page 102

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine

GetDimensionAggregate

Syntax

`GetDimensionAggregate`*`(DimName)`*

Description

Use the GetDimensionAggregate method to return the name of the user function used for the aggregate for the dimension specified by *DimName.*

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension that you want the aggregate user function.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetDimensionAggregate, page 104

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 82

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 123

GetEffects

Syntax

```
GetEffects(EffectType)
```

Description

Any cube that is affected by another cube is an *effect* of that cube. Use the GetEffects method to return a list of all the cubes affected by this cube.

The list of cubes is returned as an array of string.

Parameters

Parameter	***Description***
EffectType	Specify the type of effects you want returned.

Value	***Description***
AnalyticModel_DirectEffects	Only return cubes that are a direct effect on this cube.
AnalyticModel_AllEffects	Return all cubes that are an effect on this cube.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCauses, page 100 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCircularDeps, page 101

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine

GetDimensionNames

Syntax

GetDimensionNames()

Description

Use the GetDimensionNames method to return an array of string containing the names of all the dimensions associated with the cube.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 98 and Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 99

Chapter 8, "Array Class," page 257

GetRule

Syntax

GetRule()

Description

Use the GetRule method to get a reference to a RuleDefn object that represents the rule for this cube.

Parameters

None.

Returns

A RuleDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleDefn Class, page 129

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetRule, page 105

SetDimensionAggregate

Syntax

SetDimensionAggregate*(DimName,UserFunctionName*)

Description

Use the SetDimensionAggregate method to specify the user function to be used for the aggregate for the dimension *DimName.*

Parameters

Parameter	***Description***
DimName	Specify the name of the dimension for which you want to set the aggregate.
UserFunctionName	Specify the name of the user function to be used as the aggregate function.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionAggregate, page 101

SetRule

Syntax

`SetRule`*`(&RuleDefn)`*

Description

Use the SetRule method to specify a RuleDefn object to be used as the rule for this cube.

Parameters

Parameter	***Description***
&RuleDefn	Specify an already instantiated RuleDefn object that you want to associate with this cube.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetRule, page 103

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleDefn Class, page 129

UsesDimension

Syntax

`UsesDimension`*`(DimName)`*

Description

Use the UsesDimension method to determine if the dimension specified by *DimName* is used by this cube.

Parameters

Parameter	*Description*
DimName	Specify the name of the dimension that you want to check for.

Returns

A Boolean value: true if the dimension is used, false otherwise.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 98; Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 99 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 103

CubeDefn Class Properties

The following section discusses the CubeDefn class properties. The properties are discussed in alphabetical order.

CalcAggregates

Description

Use this property to specify if the aggregate is calculated for this cube. This property takes a Boolean value: true if the aggregate is calculated, false otherwise.

This property is read-write.

Comments

Description

Use this property to specify additional notes about the cube.

This property is read-write.

DimensionCount

Description

This property indicates the number of dimensions associated with this cube.

This property is read-only.

FormatType

Description

Use this property to specify the format of the cube. The values are:

Value	***Description***
AnalyticModel_Format_Number	The format is of type number.
AnalyticModel_Format_Date	The format is of type date.
AnalyticModel_Format_ Member	The format is of type member.
AnalyticModel_Format_Text	The format is of type text.

This property is read-write.

IsVirtual

Description

Use this property to specify if the cube is a virtual cube, that is, if it doesn't store data. This property takes a Boolean value: true if the cube is a virtual cube, false otherwise.

This property is read-write.

Name

Description

This property specifies the name of the cube.

This property is read-only.

ValueDimensionName

Description

Use this property to specify the dimension name for cubes that have a format of member.

This property is only valid when a cube has a format of member.

This property is read-write.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," FormatType, page 107

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine

CubeCollectionDefn Class

Use the CubeCollectionDefn class to access cube collections associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn class methods:

- AddCubeCollection
- CopyCubeCollection
- GetCubeCollection

CubeCollectionDefn Class Methods

The following section discusses the CubeCollectionDefn class methods. The methods are discussed in alphabetical order.

AttachCube

Syntax

`AttachCube`*`(CubeName)`*

Description

Use the AttachCube method to attach the existing cube specified by *CubeName* to the cube collection.

You should only specify cube collections comprised of derived/work records as the main record for a presentation cube collection.

Parameters

Parameter	*Description*
CubeName	Specify the name of the cube you want to attach.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachCube, page 109 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 111

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 98

DetachCube

Syntax

`DetachCube``(`*`CubeName`*`)`

Description

Use the DetachCube to detach the cube specified by *CubeName* from the cube collection.

Parameters

Parameter	*Description*
CubeName	Specify the name of the cube that you want to detach.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 98 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 111

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 98

GetAggregateMapping

Syntax

`GetAggregateMapping`*`(PartName,IsCube)`*

Description

Use the GetAggregateMapping method to return the aggregate mapping field specified by the part *PartName.*

Parameters

Parameter	***Description***
PartName	Specify the part name for which you want the aggregate mapping field name.
IsCube	Specify whether the part is a cube or not. This parameter takes a Boolean value, true if the part is a cube, false otherwise.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 113; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPersistAggregate, page 114; Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 115; Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 117 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 119

GetCubeNames

Syntax

GetCubeNames()

Description

Use the GetCubeNames method to return an array of strings containing all the names of the cubes associated with this cube collection.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 59; Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCube, page 63; Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCube, page 69; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 74 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 84

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 98

Chapter 8, "Array Class," page 257

GetDimensionNames

Syntax

GetDimensionNames()

Description

Use the GetDimensionNames method to return an array of string containing the names of all the dimensions associated with the cube collection.

The dimension names are returned in an array of string.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 98 and Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 99

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 93

GetDimSort

Syntax

`GetDimSort(DimensionName)`

Description

Use the GetDimSort method return the sorting keys used for the dimension.

The sorting keys are returned as an array of any.

The first string is either true or false. If the first string is true it indicates that the dimension is sorted by the names of the members in the dimension.

The second string then is either true or false, indicating whether the sort is ascending (or descending). If the first string is false it indicates that the dimension is sorted by keys, which are the specified by the other strings in the array. The keys are names of cubes. The strings are as follows:

- Ascend1
- CubeName1
- Ascend2
- CubeName2
- Ascend3
- CubeName3

Parameters

Parameter	***Description***
DimensionName	Specify the name of the dimension for which you want the sorting keys.

Returns

An array of any.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetDimSort, page 116

Chapter 2, "Analytic Calculation Engine Classes," GetDimSort, page 42 and Chapter 2, "Analytic Calculation Engine Classes," SetDimSort, page 47

GetFieldMapping

Syntax

`GetFieldMapping``(PartName,IsCube)`

Description

Use the GetFieldMapping method to return the name of the mapped field name for the part specified by *PartName.*

Parameters

Parameter	***Description***
PartName	Specify the name of the part for which you want the mapped field name.
IsCube	Specify whether this part is a cube or not. This parameter takes a Boolean value: true if the part is a cube, false otherwise.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetAggregateMapping, page 110; Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 117 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 115

GetFilter

Syntax

`GetFilter``(DimensionName)`

Description

Use the GetFilter method to return the name of the user function used as a filter for the dimension specified by *DimensionName.*

Parameters

Parameter	***Description***
DimensionName	Specify the name of the dimension for which you want the filter name.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 115

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Creating Rules, Formulas, and User Functions," Filter User Functions

GetPersistAggregate

Syntax

`GetPersistAggregate``(DimensionName)`

Description

Use the GetPersistAggregate method to return the value for the persist aggregate for the dimension specified by *DimensionName.*

Parameters

Parameter	***Description***
DimensionName	Specify the name of the dimension for which you want to find the persist aggregate value.

Returns

An integer, which is one of the following values:

Value	***Description***
AnalyticModel_AggrType_Root	The persist aggregate type is *Root,* that is, only the root node's data is saved to the database.
AnalyticModel_AggrType_All	The persist aggregate type is *All,* that is, save all of the dimension's aggregate data to the database.
AnalyticModel_AggrType_ None	The persist aggregate type is *None,* that is, do not save any of the dimension's aggregate data to the database.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetAggregateMapping, page 110; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 113; Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 115; Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 117 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 119

SetAggregateMapping

Syntax

`SetAggregateMapping``(`*`PartName,IsCube,AggregateFieldName`*`)`

Description

Use the SetAggregateMapping method to specify an aggregate field for the part specified by *PartName.*

You can map a field to only one data cube or dimension within one cube collection.

Parameters

Parameter	***Description***
PartName	Specify the name of the part for which you want to add an aggregate field.
IsCube	Specify whether the part is a cube. This parameter takes a Boolean value, true if the part is a cube, false otherwise.
AggregateFieldName	Specify the name of the field to be used to hold the aggregate value.

Returns

A string containing the name of the field that contains the aggregate value.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 119; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 113; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPersistAggregate, page 114; Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 117 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 119

SetDimSort

Syntax

```
SetDimSort(DimName,IsAscending [, CubeName1, IsAscending2,CubeName2,
IsAscending3,CubeName3])
```

Description

Use the SetDimSort method to specify the sorting order for the dimension.

If *CubeName1* is an empty string, the dimension is sorted by the names of the members of the dimension, and the other parameters are not used.

If *CubeName1* is not an empty string, the dimension is sorted by cubes, as specified by the other parameters.

The Boolean parameter *IsAscending* defines whether sorting should be in ascending or descending order. Up to three sort keys may be specified. *CubeName1* is the primary sort key. *CubeName2* is used to sub-sort any members that have the same key value under *CubeName1,* and so on.

Parameters

Parameter	*Description*
DimensionName	Specify the name of the dimension that you would like to specify a sort order for.
IsAscending	Specify if the dimension should be sorted in ascending or descending order. This parameter takes a Boolean value: true if the sort should be ascending, false if it should be descending.
CubeName1	Specify whether the dimension is to be sorted by members or by cubes. If the dimension is to be sorted by members, specify a null string for this parameter. If the dimension is to be sorted by cubes, specify a cube name.
IsAscending2	Specify if the sort for *CubeName1* is in ascending or descending order. This parameter takes a Boolean value: true if the sort is ascending, false if its descending.
CubeName2	Specify the name of a cube to be used to sub-sort the primary sort specified by *CubeName1*.
IsAscending3	Specify if the sort for *CubeName2* is in ascending or descending order. This parameter takes a Boolean value: true if the sort is ascending, false if its descending.
CubeName3	Specify the name of a cube to be used to sub-sort the primary sort specified by *CubeName2*.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 119

Chapter 2, "Analytic Calculation Engine Classes," GetDimSort, page 42

SetFieldMapping

Syntax

```
SetFieldMapping(PartName,FieldName,IsCube)
```

Description

Use the SetFieldMapping method to specify the name of the main field used for mapping the part specified by *PartName.*

You can map a field to only one data cube or dimension within one cube collection.

Parameters

Parameter	Description
PartName	Specify the name of the part for which you want to assign a mapping field.
FieldName	Specify the name of the field to be used for the field mapping.
IsCube	Specify whether or not the part is a cube. This parameter takes a Boolean value: true if the part is a cube, false otherwise.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetAggregateMapping, page 110; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 113; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPersistAggregate, page 114; Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 115 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 119

SetFilter

Syntax

```
SetFilter(DimensionName,UserFunctionName)
```

Description

Use the SetFilter method to specify the user function to be used as a filter for the dimension specified by *DimensionName.*

Parameters

Parameter	*Description*
DimensionName	Specify the name of the dimension for which you want to set a filter.
UserFunctionName	Specify the name of a user function to be used as a filter for this dimension.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFilter, page 114

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Creating Rules, Formulas, and User Functions," Filter User Functions

SetPersistAggregate

Syntax

`SetPersistAggregate``(`*`DimensionName,AggregateType`*`)`

Description

Use the SetPersistAggregate to specify the value for the persist aggregate for the dimension specified by *DimensionName.*

Parameters

Parameter	*Description*
DimensionName	Specify the name of the dimension for which you want to set the persist aggregate.
AggregateType	Specify the aggregate type. Values are:

Value	*Description*
AnalyticModel_AggrType_Root	The persist aggregate type is *Root*, that is, only the root node's data is saved to the database.

Value	Description
AnalyticModel_AggrType_All	The persist aggregate type is *All*, that is, save all of the dimension's aggregate data to the database.
AnalyticModel_AggrType_ None	The persist aggregate type is *None*, that is, do not save any of the dimension's aggregate data to the database.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetAggregateMapping, page 110; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 113; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPersistAggregate, page 114; Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 115 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 117

UsesCube

Syntax

`UsesCube``(`*`CubeName`*`)`

Description

Use the UsesCube method to determine if the cube collection uses the cube specified by *CubeName.*

Parameters

Parameter	Description
CubeName	Specify the name of the cube that you want to verify.

Returns

A Boolean value, true if the specified cube is part of the cube collection, false otherwise.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachCube, page 108; Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachCube, page 109 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 111

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 98

UsesDimension

Syntax

`UsesDimension``(`*`DimensionName`*`)`

Description

Use the UsesDimension method to determine if the cube collection uses the dimension specified by *DimensionName.*

Parameters

Parameter	*Description*
DimensionName	Specify the name of the dimension that you want to verify.

Returns

A Boolean value: true if the dimension is part of the cube collection, false otherwise.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 98

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 93

CubeCollectionDefn Class Properties

The following section discusses the CubeCollectionDefn class properties. The properties are discussed in alphabetical order.

AggregateRecName

Description

Use this property to specify the name of the aggregate record associated with this cube collection. This property takes a string value.

This property is read-write.

Comments

Description

Use this property to specify additional notes about the cube collection.

This property is read-write.

CubeCount

Description

This property returns the number of cubes in the cube collection.

This property is read-only.

DimensionCount

Description

This property returns the number of dimensions associated with this cube collection.

This property is read-only.

Name

Description

This property returns the name of the cube collection as a string.

This property is read-only.

RecordName

Description

Use this property to specify the name of the main record associated with the cube collection. This property takes a string value.

This property is read-write.

UserFunctionDefn Class

Use the UserFunctionDefn class to access the user functions that are associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn class methods:

- AddUserFunction
- CopyUserFunction
- GetUserFunction

UserFunctionDefn Class Methods

In this section we discuss the UserFunctionDefn class methods. They are described in alphabetical order.

GetRule

Syntax

`GetRule``()`

Description

Use the GetRule method to return a RuleDefn object that represents the rule associated with this user function.

Parameters

None.

Returns

A RuleDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleDefn Class, page 129

SetRule

Syntax

`SetRule`_`(&Rule)`_

Description

Use the SetRule method to specify a rule defined by a RuleDefn object to be the rule for this user function.

Parameters

Parameter	*Description*
&Rule	Specify an already instantiated RuleDefn object that you want to use as the rule for this user function.

Returns

None.

UserFunctionDefn Class Properties

The following section discusses the UserFunctionDefn class properties. The properties are discussed in alphabetical order.

Comments

Description

Use this property to specify additional notes about the user function.

This property is read-write.

Name

Description

This property specifies the name of the user function.

This property is read-only.

OrganizerDefn Class

Use the OrganizerDefn class to access the organizers that are associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn class methods:

- AddOrganizer
- CopyOrganizer
- GetOrganizer

OrganizerDefn Class Methods

The following section discusses the OrganizerDefn class methods. The methods are discussed in alphabetical order.

AttachPart

Syntax

`AttachPart``(`*`PartName,PartType`*`)`

Description

Use the AttachPart method to attach the part specified by *PartName* to the organizer. The part must exist in the analytic model.

Parameters

Parameter	Description
PartName	Specify the name of the part that you want to attach.
PartType	Specify the type of part that you want to attach. Values are:

Value	Description
AnalyticModel_Dimension	Attach a dimension.
AnalyticModel_Cube	Attach a cube.
AnalyticModel_ CubeCollection	Attach a cube collection.
AnalyticModel_UserFunction	Attach a user function.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachPart, page 126; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPartNames, page 127 and Chapter 3, "Analytic Calculation Engine Metadata Classes," UsesPart, page 128

DetachPart

Syntax

`DetachPart`*`(PartName,PartType)`*

Description

Use the DetachPart method to detach the part specified by *PartName* from the organizer.

Parameters

Parameter	Description
PartName	Specify the name of the part that you want to detach.

Parameter	Description
PartType	Specify the type of the part that you want to detach. Valid values are:

Value	Description
AnalyticModel_Dimension	Detach a dimension.
AnalyticModel_Cube	Detach a cube.
AnalyticModel_ CubeCollection	Detach a cube collection.
AnalyticModel_UserFunction	Detach a user function.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachPart, page 125; Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPartNames, page 127 and Chapter 3, "Analytic Calculation Engine Metadata Classes," UsesPart, page 128

GetPartNames

Syntax

```
GetPartNames()
```

Description

Use the GetPartNames method to return a list of the names of the parts used by the organizer. The names are returned as an array of string.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachPart, page 125; Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachPart, page 126 and Chapter 3, "Analytic Calculation Engine Metadata Classes," UsesPart, page 128

UsesPart

Syntax

`UsesPart(PartName,PartType)`

Description

Use the UsesPart method to determine if the organizer uses the part specified by *PartName.*

Parameters

Parameter	Description
PartName	Specify the name of the part that you want to verify.
PartType	Specify the type of the part that you want to verify. Valid values are:

Value	Description
AnalyticModel_Dimension	A dimension.
AnalyticModel_Cube	A cube.
AnalyticModel_ CubeCollection	A cube collection.
AnalyticModel_UserFunction	A user function.

Returns

A Boolean value: true if the part is associated with the organizer, false otherwise.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachPart, page 125; Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachPart, page 126 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPartNames, page 127

OrganizerDefn Class Properties

The following section discusses the OrganizerDefn class properties. The properties are discussed in alphabetical order.

Comments

Description

Use this property to specify additional notes about the organizer.

This property is read-write.

Name

Description

This property specifies the name of the organizer.

This property is read-only.

RuleDefn Class

Use the RuleDefn class to access a rule that is associated with an analytic model. You instantiate an object of this class using the GetRule method for both the CubeDefn and UserFunctionDefn classes.

RuleDefn Class Methods

In this section, we discuss the RuleDefn class methods. The methods are described in alphabetical order.

AddRuleExpression

Syntax

`AddRuleExpression`*`(&Expr)`*

Description

Use the AddRuleExpression method to add a rule expression to the rule. Use the classes in the RuleExpression subpackage to create the expression specified by *&Expr*.

After you have finished adding rule expressions to the rule, you need to use the GenerateRule method to generate the rule.

Parameters

Parameter	***Description***
&Expr	Specify the rule expression that you want added to the rule definition.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GenerateRule, page 130

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleExpressions Classes, page 131

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to create the rule string for the rule that represents the current rule expressions that have been added using AddRuleExpression.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddRuleExpression, page 129

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleExpressions Classes, page 131

RuleDefn Class Property

The following describes the RuleDefn class property.

RuleString

Description

This property specifies a string representation of the rule.

After the rule is returned using the GetRule method, this string represents the existing rule cube or user function.

After AddRuleExpression and GenerateRule are called, this is the string representation of the rule.

This property is read-only.

RuleExpressions Classes

The RuleExpressions classes represents the different parts of the analytic calculation engine rule grammar. These objects are created using the Create built-in function and the constructor for that class, then added to the RuleDefn object with the AddRuleExpression method.

The following are the RuleExpressions classes:

- Assignment
- Comparison
- Constant
- Constants
- Cube
- ExpressionBlock
- FunctionCall
- MemberReference

- Operation
- RuleExpression
- Variable

Note. The RuleExpression object is the object that the other objects are derived from. You should not create or use this object directly.

Using the Constants Class

All of the constants used with the RuleExpression classes, such as FunctionCall, comparison, operation, and so on, are actually properties of the constants class. You must always instantiate an object of the constants class to use any constants in your program.

For example, the comparison class uses a constant to test whether two operands are equal. Use the following code to create a comparison testing this:

```
&Comparison = create Comparison(&Constants.Comparison_Type_Equal);
```

All of the properties for the constants class are listed with the classes that use them.

Assignment Class

An assignment object represents an assignment statement in an analytic calculation rule.

Use the following to create an assignment object:

```
&Assignment = create Assignment();
```

Assignment Class Method

The following is the method for the assignment class.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddRuleExpression, page 129

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleExpressions Classes, page 131

Assignment Class Properties

In this section we discuss the assignment class properties. The properties are described in alphabetical order.

Expression

Description

Use the Expression property to specify the right-hand side of the assignment statement.

This property is read-write.

Variable

Description

Use the Variable property to specify the left-hand side of the assignment statement.

This property is read-write.

Comparison Class

A comparison object represents a comparison statement in an analytic calculation rule.

Use the following to create a comparison class object:

```
&Comparison = create Comparison(&Constants.Comparison_Type);
```

Where *Comparison_Type* is one of the following:

Value	***Description***
Comparison_Type_Equal	Compare if the value of Operand1 equals Operand2.
Comparison_Type_Greater	Compare if the value of Operand1 is greater than the value of Operand2.
Comparison_Type_Less	Compare if the value of Operand1 is less than the value of Operand2.
Comparison_Type_GreaterEq	Compare if the value of Operand1 is greater or equal to the value of Operand2.
Comparison_Type_LessEq	Compare if the value of Operand1 is less than or equal to the value of Operand2.
Comparison_Type_NotEq	Compare if the value of Operand1 is not equal to the value of Operand2.

Specify Operand1 and Operand2 with the Operand1 and Operand2 comparison class properties.

The following code example creates a rule that compares if the first operand is greater than or equal to the second operand, then adds the rule using the AddArgument method.

```
&Comparison = create Comparison(&Constants.Comparison_Type_GreaterEq);
&Constant = create Constant(&Constants.Constant_Type_Literal, "1000");
&Comparison.Operand1 = &Constant;
&Constant = create Constant(&Constants.Constant_Type_Literal, "100");
&Comparison.Operand2 = &Constant;
&FunCall.AddArgument(&Comparison);
```

Comparison Class Method

The following is the comparison class method.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddRuleExpression, page 129

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleExpressions Classes, page 131

Comparison Class Properties

In this section we discuss the comparison class properties. The properties are described in alphabetical order.

Operand1

Description

Use this property to specify the RuleExpression object that is the first operand to be used in the comparison.

This property is read-write.

Operand2

Description

Use this property to specify the RuleExpression object that is the second operand to be used in the comparison.

This property is read-write.

Type

Description

This property returns the type of the comparison that was used to instantiate the comparison object.

The values are:

Value	*Description*
Comparison_Type_Equal	Compare if the value of Operand1 equals Operand2.
Comparison_Type_Greater	Compare if the value of Operand1 is greater than the value of Operand2.
Comparison_Type_Less	Compare if the value of Operand1 is less than the value of Operand2.
Comparison_Type_GreaterEq	Compare if the value of Operand1 is greater than or equal to the value of Operand2.
Comparison_Type_LessEq	Compare if the value of Operand1 is less than or equal to the value of Operand2.
Comparison_Type_NotEq	Compare if the value of Operand1 is not equal to the value of Operand1.

This property is read-only.

Constant Class

A constant object represents a constant statement in an analytic calculation rule.

Note. This is not the same as the constants class.

A constant object exposes all the constants that are passed to the constructors of the various RuleExpression objects.

Use the following to create a constant class object:

```
&Constant = create Constant(&Constants.Constant_Type, [&Constants.]Constant_Value);
```

where *Constant_Type* is one of the following:

Value	*Description*
Constant_Type_Builtin	The *Constant_Value*is an analytic calculation engine built-in function. In this instance, the constant value must be prefaced with an already instantiated constants object. For possible values, see below.
Constant_Type_Number	The *Constant_Value* is a number.

Value	***Description***
Constant_Type_Literal	The *Constant_Value* is a string.

When the *Constant_Type* is Constant_Type_Builtin, the *Constant_Value* is one of the following:

- Constant_Builtin_ALL
- Constant_Builtin_BASE_E
- Constant_Builtin_DEFAULT
- Constant_Builtin_DETAILS
- Constant_Builtin_DIRECT
- Constant_Builtin_FALSE
- Constant_Builtin_FORWARD
- Constant_Builtin_PI
- Constant_Builtin_REVERSE
- Constant_Builtin_TRUE

The following code creates a PeopleSoft Analytic Calculation Engine built-in function to be used as a constant:

```
   &Constant = create Constant(&Constants.Constant_Type_Builtin,⇒
 &Constants.Constant_Builtin_REVERSE);
```

The following code creates a literal (string) constant:

```
   &Constant = create Constant(&Constants.Constant_Type_Literal, "GENERAL");
```

The following code creates a number constant:

```
&Constant = create Constant(&Constants.Constant_Type_Number, "-1");
```

Constant Class Method

The following is the constant class method.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

Constant Class Properties

In this section we discuss the constant class properties. The properties are described in alphabetical order.

Type

Description

This property returns the value of the *Constant Type* that was used to instantiate the constant class object.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Constant Class, page 136

Value

Description

This property returns the value of the *Constant Value* that was used to instantiate the constant class object.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Constant Class, page 136

Constants Class

All of the constants used with the RuleExpression classes, such as FunctionCall, comparison, operation, and so on, are actually properties of the constants class. You must always instantiate an object of the constants class to use any constants in your program.

For example, the comparison class uses a constant to test whether two operands are equal. Use the following code to create a comparison testing this:

```
&Comparison = create Comparison(&Constants.Comparison_Type_Equal);
```

All of the properties for the constants class are listed with the classes that use them.

Cube Class

A cube object represents a cube statement in an analytic calculate engine rule.

Use the following to create a cube object.

```
&Assignment = create Cube("CubeName");
```

Cube Class Methods

In the following section, we discuss the cube class methods. The methods are described in alphabetical order.

AddIndex

Syntax

AddIndex*(&MemberReference)*

Description

Use the AddIndex method to add a MemberReference object to the cube.

Parameters

Parameter	*Description*
&MemberReference	Specify an already instantiated MemberReference object to be added to the cube.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," MemberReference Class, page 149

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

GetIndexes

Syntax

GetIndexes()

Description

Use the GetIndexes method to return an array of MemberReference's (indexes) for this cube. These MemberReferences are the MemberReferences that were added with the AddIndex method.

Parameters

None.

Returns

An array of MemberReference objects.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," MemberReference Class, page 149

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddIndex, page 139

Cube Class Property

The following is the cube class property.

Name

Description

This property indicates the name of the cube that was used to instantiate the cube object.

This property is read-only.

ExpressionBlock Class

For some function, you need multi-statement nested expressions. You need to use the ExpressionBlock class to group these expressions. For example, all the statements inside of a FOR statement should be included in an expression block.

```
FOR(&Index, 1, PERIOD,
   SET(&Value, &Value + 1);
   SET(&countPl4, &countPl4 + 4);
);
```

Use the following code to create an ExpressionBlock object:

```
&ExpressionBlock = create ExpressionBlock();
```

ExpressionBlock Methods

In the following section we discuss the ExpressionBlock class methods. The methods are described in alphabetical order.

AddRuleExpression

Syntax

`AddRuleExpression`*`(&Expr)`*

Description

Use the AddRuleExpression method to add an expression to the expression block and the rule. Use the classes in the RuleExpression subpackage to create the expression specified by *&Expr*.

Note. As the rule string is generated, the system adds a ";" after each RuleExpression, except the last.

Parameters

Parameter	*Description*
&Expr	Specify the rule expression that you want added to the expression block.

Returns

None.

GetRuleExpressions

Syntax

`GetRuleExpressions``()`

Description

Use the GetRuleExpressions to return an array of rule expression objects that have been added using the AddRuleExpression method.

Parameters

None.

Returns

An array of RuleExpression objects.

FunctionCall Class

A FunctionCall object represents a function call statement in an analytic calculation engine rule.

Use the following code to create a FunctionCall object:

```
&Assignment = create FunctionCall(&Constants.Function_Call_Type, [&Constants.]⇒
Function_Call_Name);
```

Where *Function_Call_Type* can be one of the following:

Value	***Description***
FunctionCall_Type_Builtin	Specifies that the function call is a PeopleSoft Analytic Calculation Engine built-in functions. In this instance, the function name must be prefaced with an already instantiated constants object. For the built-in function values used with *Function_Call_Name,* see below.
FunctionCall_Type_UserFunc	Specifies that the function call is a user function. The value specified for *Function_Call_Name* must be an already created user function defined in this analytic model. Specify this value as a string.

When the *Function_Call_Type* is specified as Function_Call_Type_Builtin, the *Function_Call_Name* must be one of the following:

- FunctionCall_Builtin_ABS
- FunctionCall_Builtin_ACOS
- FunctionCall_Builtin_ARGUMENTS
- FunctionCall_Builtin_ASC
- FunctionCall_Builtin_ASIN
- FunctionCall_Builtin_AT
- FunctionCall_Builtin_ATAN
- FunctionCall_Builtin_BREAK
- FunctionCall_Builtin_CASE

- FunctionCall_Builtin_CHANGE
- FunctionCall_Builtin_CHILDCOUNT
- FunctionCall_Builtin_CHR
- FunctionCall_Builtin_CONSOL
- FunctionCall_Builtin_COS
- FunctionCall_Builtin_CUBEID
- FunctionCall_Builtin_CUBSUM
- FunctionCall_Builtin_CUMAVG
- FunctionCall_Builtin_DAVG
- FunctionCall_Builtin_DAY
- FunctionCall_Builtin_DCOUNT
- FunctionCall_Builtin_DDB
- FunctionCall_Builtin_DEC
- FunctionCall_Builtin_DLOOKUP
- FunctionCall_Builtin_DMAX
- FunctionCall_Builtin_DMIN
- FunctionCall_Builtin_DSUM
- FunctionCall_Builtin_E
- FunctionCall_Builtin_FIRST
- FunctionCall_Builtin_FOR
- FunctionCall_Builtin_FORCHILDREN
- FunctionCall_Builtin_FORMEMBERS
- FunctionCall_Builtin_FV
- FunctionCall_Builtin_GROUPAVG
- FunctionCall_Builtin_GROUPBY
- FunctionCall_Builtin_GROUPMAX
- FunctionCall_Builtin_GROUPMIN
- FunctionCall_Builtin_GROUPSUM
- FunctionCall_Builtin_GROW
- FunctionCall_Builtin_IF

- FunctionCall_Builtin_IFNPV
- FunctionCall_Builtin_INC
- FunctionCall_Builtin_INDICATE
- FunctionCall_Builtin_INPUT
- FunctionCall_Builtin_INSUBTREE
- FunctionCall_Builtin_INTERCEPT
- FunctionCall_Builtin_IRR
- FunctionCall_Builtin_ISINPUT
- FunctionCall_Builtin_LEFT
- FunctionCall_Builtin_LEN
- FunctionCall_Builtin_LN
- FunctionCall_Builtin_LOWER
- FunctionCall_Builtin_MATCH
- FunctionCall_Builtin_MAX
- FunctionCall_Builtin_MBR2TEXT
- FunctionCall_Builtin_MEDIAN
- FunctionCall_Builtin_MEMBER
- FunctionCall_Builtin_MID
- FunctionCall_Builtin_MIN
- FunctionCall_Builtin_MOD
- FunctionCall_Builtin_MONTH
- FunctionCall_Builtin_NEXT
- FunctionCall_Builtin_NPER
- FunctionCall_Builtin_NUM2TEXT
- FunctionCall_Builtin_NUMMEMBERS
- FunctionCall_Builtin_NPV
- FunctionCall_Builtin_PARAMETER
- FunctionCall_Builtin_PARENT
- FunctionCall_Builtin_PCT
- FunctionCall_Builtin_PERCENTILE

- FunctionCall_Builtin_PI
- FunctionCall_Builtin_PMT
- FunctionCall_Builtin_PREV
- FunctionCall_Builtin_PREVSELF
- FunctionCall_Builtin_PV
- FunctionCall_Builtin_QUARTILE
- FunctionCall_Builtin_RAND
- FunctionCall_Builtin_RATE
- FunctionCall_Builtin_REPLACE
- FunctionCall_Builtin_RETURN
- FunctionCall_Builtin_RIGHT
- FunctionCall_Builtin_ROUND
- FunctionCall_Builtin_SELF
- FunctionCall_Builtin_SET
- FunctionCall_Builtin_SIN
- FunctionCall_Builtin_SLN
- FunctionCall_Builtin_SLOPE
- FunctionCall_Builtin_SQRT
- FunctionCall_Builtin_STDEV
- FunctionCall_Builtin_SYD
- FunctionCall_Builtin_TAN
- FunctionCall_Builtin_TEXT2MBR
- FunctionCall_Builtin_TEXT2NUM
- FunctionCall_Builtin_THIS
- FunctionCall_Builtin_THISCUBE
- FunctionCall_Builtin_TRUNC
- FunctionCall_Builtin_UPPER
- FunctionCall_Builtin_VAR
- FunctionCall_Builtin_WHILE
- FunctionCall_Builtin_YEAR

The following code creates a function call using a PeopleSoft Analytic Calculation Engine built-in function.

```
   &FunCall = create FunctionCall(&Constants.FuncCall_Type_Builtin,⇒
 &Constants.FuncCall_Builtin_ABS);
```

See Also

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Using Built-in Functions in Analytic Models"

FunctionCall Class Methods

In this section we discuss the FunctionCall class methods. The methods are described in alphabetical order.

AddArgument

Syntax

AddArgument*(&RuleExpression)*

Description

Use the AddArgument method to add an argument to this function call.

Parameters

Parameter	***Description***
&RuleExpression	Specify an already instantiated rule expression object, such as a cube object.

Returns

None.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

GetArguments

Syntax

`GetArguments``()`

Description

Use the GetArguments method to return an array of RuleExpression objects associated with this FunctionCall. These are the RuleExpression objects that were added using the AddArgument method.

Parameters

None.

Returns

An array of RuleExpression objects.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddArgument, page 147

FunctionCall Class Properties

In this section we discuss the FunctionCall class properties. The properties are described in alphabetical order.

Name

Description

This property specifies the name of the function call used to instantiate the FunctionCall object.

This property is read-only.

Type

Description

This property specifies the type of the function call used to instantiate the FunctionCall object.

This property is read-only.

MemberReference Class

A MemberReference object represents a member in an analytic calculation engine rule.

Use the following code to create a MemberReference object:

```
&Member = create MemberReference(DimensionName, MemberName);
```

MemberReference Class Method

This section discusses the MemberReference class method.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

MemberReference Class Properties

In this section we discuss MemberReference class properties. The properties are described in alphabetical order.

Dimension

Description

This property specifies the name of the dimension used to create the MemberReference object.

This property is read-only.

Member

Description

This property specifies the name of the member used to create the MemberReference object.

This property is read-only.

Operation Class

An operation object represents an operation in an analytic calculation engine rule. The operation is generally some type of mathematical function performed between two operands. Specify the operands for the operation using the Operand1 and Operand2 operation class properties.

Use the following code to create an operation object:

```
&Comparison = create Operation(&Constants.Operation_Type);
```

Where *Operation_Type* has a value of one of the following:

Value	***Description***
Operation_Type_Plus	The operation is addition.
Operation_Type_Minus	The operation is subtraction.
Operation_Type_Mult	The operation is multiplication.
Operation_Type_Div	The operation is division.
Operation_Type_And	Use a logical AND with the operation.
Operation_Type_Or	Use a logical OR with the operation.
Operation_Type_Not	Use a logical NOT with the operation.
Operation_Type_Neg	Use a mathmatical negate with the operation.
Operation_Type_Exp	Use a mathmatical exponent with the operation.

The following code creates an operations that is a multiplication between two operands.

```
&operation = create Operation(&Constants.Operation_Type_Mult);
&Constant = create Constant(&Constants.Constant_Type_Literal, "PI");
&operation.Operand1 = &Constant;
&Constant = create Constant(&Constants.Constant_Type_Literal, "SLOPE");
&operation.Operand2 = &Constant;
&FunCall.AddArgument(&operation);
```

Operation Class Method

The following is the operation class method.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

Operation Class Properties

In this section we discuss the operation class properties. The properties are described in alphabetical order.

Operand1

Description

Use this property to specify the RuleExpression object that is the first operand to be used in the operation.

This property is read-write.

Operand2

Description

Use this property to specify the RuleExpression object that is the second operand to be used in the operation.

This property is read-write.

Type

Description

This property returns the type of the operation used to instantiate the operation object.

The values for this property are:

Value	***Description***
Operation_Type_Plus	The operation is addition.
Operation_Type_Minus	The operation is subtraction.
Operation_Type_Mult	The operation is multiplication.
Operation_Type_Div	The operation is division.
Operation_Type_And	Use a logical AND for the operation.
Operation_Type_OR	Use a logical OR for the operation.
Operation_Type_Not	Use a logical NOT for the operation.
Operation_Type_Neg	Use a mathmatical negate for the operation.
Operation_Type_Exp	Use a mathmatical exponent for the operation.

Variable Class

A variable object represents a variable in an analytic calculation engine rule.

Use the following code to create a variable object:

```
&Variable = create Variable(&Constants.Variable_Type, "Variable_Name");
```

Where *Variable_Type* is one of the following:

Value	***Description***
Variable_Type_Auto	Use a local variable.
Variable_Type_Expression	The variable is in an expression. Specify an already instantiated rule expression object for the *Variable_Name.*
Variable_Type_Dimension	The variable is a dimension name. Specify a dimension name for the *Variable_Name.*

The following code example creates a variable, then adds it to the rule.

```
&Variable = create Variable(&Constants.Variable_Type_Auto, "StartVal");
&FunCall.AddArgument(&Variable);
```

Variable Class Method

The following is the variable class method.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

Variable Class Properties

In this section we discuss the variable class properties. The properties are described in alphabetical order.

Name

Description

This property specifies the name of the variable that was used to create the variable object, passed in the constructor.

This property is read-only.

Type

Description

This property specifies the type of the variable, passed in the constructor. The possible values are:

Value	***Description***
Variable_Type_Auto	A local variable.
Variable_Type_Expression	The variable is an expression. Specify an already instantiated rule expression object for the *Variable_Name.*
Variable_Type_Dimension	The variable is a dimension name. Specify a dimension name for the *Variable_Name.*

This property is read-only.

Analytic Model Metadata Classes Examples

The following are some examples of the most general cases of using the Analytic Model Metadata classes.

Creating an Analytic Model Code Example

The following code creates an analytic model definition "from scratch" without retrieving an existing definition from the Application Designer, adds parts such as dimensions, cubes, and cube collections, then saves the definition.

```
import PT_ANALYTICMODELDEFN:*;

Function CreateGENXModel

   Local PT_ANALYTICMODELDEFN:AnalyticModelDefn &Model;
   Local PT_ANALYTICMODELDEFN:UserFunctionDefn &UserFunc;
   Local PT_ANALYTICMODELDEFN:DimensionDefn &Dim;
   Local PT_ANALYTICMODELDEFN:CubeDefn &Cube;
   Local PT_ANALYTICMODELDEFN:CubeCollectionDefn &CubeColl;
   Local array of string &arr;

   &ACEMODELID = QE_ACE_META_WK.QE_ACE_MODELID.Value;

   &Model = create PT_ANALYTICMODELDEFN:AnalyticModelDefn(&ACEMODELID);

   &Model.Create();

   /* Model Properties */


   &Model.Description = QE_ACE_META_WK.DESCR;
   &Model.LongDescription = QE_ACE_META_WK.DESCR200;
   &Model.MaxDelta = QE_ACE_META_WK.QE_ACE_MAXDELTA_FL;
   &Model.MaxIterations = QE_ACE_META_WK.QE_ACE_MAXITER_FLD;

   If (QE_ACE_META_WK.QE_ACE_CIRCWARN_FL.Value = 0) Then
      &Circ = True;
   Else
      &Circ = False;
   End-If;

   If (QE_ACE_META_WK.QE_ACE_RESOLVE_FLD.Value = 0) Then
      &Resolve = True;
   Else
      &Resolve = False;
   End-If;

   &Model.ResolveCircularDeps = &Resolve;
   &Model.CircularFormulaWarn = &Circ;


   /* Add User Functions */

   &UserFunction = &Model.AddUserFunction("FILTERPRODUCTS");
   &Rule = &UserFunction.GetRule();
   &Rule.RuleString = "IF( ((UNIT_COST = 0) .AND. (UNITS_SOLD = 0) .AND. (PROD_⇒
SALES =0 )) , RETURN(0), RETURN(1))";
   &UserFunction.SetRule(&Rule);


   /* Add the Dimensions */
   &Dim = &Model.AddDimension("MONTH");

   &Dim = &Model.AddDimension("PRODUCTS");
   &Dim.TotalMemberName = "TOTAL";

   &Dim = &Model.AddDimension("PROD_CAT");

   &Dim = &Model.AddDimension("REGION");

   /* Add the cubes */
   &Cube = &Model.AddCube("PROD_SALES");
   &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
   &Cube.IsVirtual = False;
```



```
    &Cube.CalcAggregates = False;
    &Cube.Rule = "UNIT_COST * UNITS_SOLD";
    &Cube.AttachDimension("MONTH");
    &Cube.AttachDimension("PRODUCTS");
    &Cube.AttachDimension("REGION");

    &Cube = &Model.AddCube("SALES");
    &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
    &Cube.IsVirtual = False;
    &Cube.CalcAggregates = False;

    &Cube = &Model.AddCube("TGT_COST");
    &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
    &Cube.IsVirtual = False;
    &Cube.CalcAggregates = False;
    &Cube.AttachDimension("PRODUCTS");

    &Cube = &Model.AddCube("UNITS_SOLD");
    &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
    &Cube.IsVirtual = False;
    &Cube.CalcAggregates = False;
    &Cube.AttachDimension("MONTH");
    &Cube.AttachDimension("PRODUCTS");
    &Cube.AttachDimension("REGION");

    &Cube = &Model.AddCube("UNIT_COST");
    &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
    &Cube.IsVirtual = False;
    &Cube.CalcAggregates = False;
    &Cube.AttachDimension("MONTH");
    &Cube.AttachDimension("PRODUCTS");
    &Cube.AttachDimension("REGION");

    /* Add Cube Collections */
    &CubeColl = &Model.AddCubeCollection("REG_SALES_IN");
    &CubeColl.RecordName = "QE_BAM_FACT_TBL";
    &CubeColl.AttachCube("PROD_SALES");
    &CubeColl.AttachCube("UNITS_SOLD");
    &CubeColl.AttachCube("UNIT_COST");
    &CubeColl.SetFieldMapping("MONTH", "QE_BAM_MONTH_FLD", False);
    &CubeColl.SetPersistAggregate("MONTH", &CubeColl.AnalyticModel_AggrType_None);
    &CubeColl.SetDimSort("MONTH", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("PRODUCTS", "QE_BAM_PRODUCT_FLD", False);
    &CubeColl.SetPersistAggregate("PRODUCTS", &CubeColl.AnalyticModel_AggrType_⇒
None);
    &CubeColl.SetDimSort("PRODUCTS", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("REGION", "QE_BAM_REGION_FLD", False);
    &CubeColl.SetPersistAggregate("REGION", &CubeColl.AnalyticModel_AggrType_None);
    &CubeColl.SetDimSort("REGION", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("PROD_SALES", "QE_BAM_PRDSALES_FL", True);
    &CubeColl.SetFieldMapping("UNITS_SOLD", "QE_BAM_SALES_FLD", True);
    &CubeColl.SetFieldMapping("UNIT_COST", "QE_BAM_UNIT_FLD", True);

    &CubeColl = &Model.AddCubeCollection("REG_SALES_PROD");
    &CubeColl.RecordName = "QE_BAM_CCSMOKE";
    &CubeColl.AttachCube("PROD_SALES");
    &CubeColl.AttachCube("UNITS_SOLD");
    &CubeColl.AttachCube("UNIT_COST");
    &CubeColl.SetFieldMapping("MONTH", "QE_BAM_MONTH_FLD", False);
    &CubeColl.SetPersistAggregate("MONTH", &CubeColl.AnalyticModel_AggrType_None);
    &CubeColl.SetDimSort("MONTH", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("PRODUCTS", "QE_BAM_PRODUCT_FLD", False);
    &CubeColl.SetPersistAggregate("PRODUCTS", &CubeColl.AnalyticModel_AggrType_⇒
```

```
None);
   &CubeColl.SetFilter("PRODUCTS", "FILTERPRODUCTS");
   &CubeColl.SetDimSort("PRODUCTS", False, "", False, "", False, "");
   &CubeColl.SetFieldMapping("REGION", "QE_BAM_REGION_FLD", False);
   &CubeColl.SetPersistAggregate("REGION", &CubeColl.AnalyticModel_AggrType_None);
   &CubeColl.SetDimSort("REGION", False, "", False, "", False, "");
   &CubeColl.SetFieldMapping("PROD_SALES", "QE_BAM_PRDSALES_FL", True);
   &CubeColl.SetFieldMapping("UNITS_SOLD", "QE_BAM_SALES_FLD", True);
   &CubeColl.SetFieldMapping("UNIT_COST", "QE_BAM_UNIT_FLD", True);

   &CubeColl = &Model.AddCubeCollection("TGT_COST_PROD");
   &CubeColl.RecordName = "QE_BAM_CC_TRGT";
   &CubeColl.AttachCube("TGT_COST");
   &CubeColl.SetFieldMapping("PRODUCTS", "QE_BAM_PRODUCT_FLD", False);
   &CubeColl.SetPersistAggregate("PRODUCTS", &CubeColl.AnalyticModel_AggrType_⇒
None);
   &CubeColl.SetDimSort("PRODUCTS", False, "", False, "", False, "");
   &CubeColl.SetFieldMapping("TGT_COST", "QE_BAM_TARGET_FLD", True);

   &Model.Save();
   /* &Valid = &Model.Validate(); */

   QE_ACE_META_WK.QE_BAM_PCSTATUS = "The Model " | &ACEMODELID | " was created and⇒
 saved.";

End-Function;
```

Chapter 4

Analytic Grid Classes

This chapter provides an overview of Analytic Calculation Engine classes, and discusses:

- The analytic grid in PeopleCode.
- Using freeze column mode.
- Error handling.
- Data types of analytic grid objects.
- Scope of analytic grid objects.
- AnalyticGrid class reference.

Important! The analytic grid classes are not supported on IBM z/OS and Linux for IBM System z platforms.

Understanding the Analytic Calculation Engine Classes

PeopleSoft Analytic Calculation Engine comprises a calculation engine plus several PeopleTools features which enable application developers to define both the calculation rules and the display of calculated data within PeopleSoft applications for the purposes of multi-dimensional reporting, data editing, and analysis.

More specifically, developers create *analytic models* in order to define the rules which are used to calculate data. Developers also create PeopleSoft Pure Internet Architecture pages with *analytic grids* in order to display the data within PeopleSoft applications. End users view, analyze, and make changes to this data. When end users save their changes, PeopleSoft Analytic Calculation Engine recalculates the data and saves the calculated data to the database.

PeopleCode enables developers to manipulate analytic calculation data as follows:

- Use the Analytic Calculation Engine classes to either retrieve or specify data in an instance of an analytic model loaded into the system, and also to calculate (or recalculate) cube values.
- Use the Analytic Calculation Engine metadata classes to manipulate an analytic model definition. For example, you can add cubes to a cube collection or rename an existing user function for a model.
- Use the Analytic Grid classes to manipulate the display of analytic calculation data on a page.
- Use the Analytic Type classes to manipulate an analytic type definition. For example, you can specify a new analytic model for an analytic type definition.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," page 53

Chapter 2, "Analytic Calculation Engine Classes," page 13

Using the Analytic Grid in PeopleCode

The PeopleCode AnalyticGrid object is a reference to a page runtime object for the analytic grid. These particular page runtime objects aren't present until the component is started.

Note. PeopleSoft builds a page grid one row at a time. Because the AnalyticGrid class applies to a complete grid, you can't attach PeopleCode that uses the AnalyticGrid class to events that occur before the grid is built; the earliest event you can use is the page Activate Event.

If you're using the analytic grid within a secondary page, the runtime object for the grid isn't created until the secondary page is run. The grid object can't be obtained until then, which means that the earliest PeopleCode event you can use to activate a grid that's on a secondary page is the Activate event for the secondary page.

The attributes you set for displaying an analytic grid remain in effect only while the page is active. When you switch between pages in a component, you have to reapply those changes *every time* the page is displayed.

In addition, the Activate event associated with a page fires every time the page is displayed. Any PeopleCode associated with that Activate event runs, which may undo the changes you made when the page was last active. For example, if you hide a grid column in the Activate event, then display it as part of a user action, when the user tabs to another page in the component, then tabs back, the Activate event runs again, hiding the grid column again.

If a user at runtime hides a column of a grid, tabs to another page in the component, then tabs back to the first page, the page is refreshed and the grid column is displayed again.

You can use the rowset class methods and properties on analytic grid data. You can access the data loaded by the analytic grid by accessing a rowset object after the grid is populated.

Use the analytic grid classes to manipulate the display of an analytic grid—that is, one associated with PeopleSoft Analytic Calculation Engine data. If you want to manipulate a grid control, use the grid classes.

See Also

Chapter 20, "Grid Classes," page 1035

Chapter 25, "Message Classes," GetRowset, page 1278

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Activate Event

Using Freeze Column Mode

If you specify Freeze Column Mode for an analytic grid in Application Designer, the analytic grid isn't populated from the component buffer by default. It is your responsibility to write your application code so that it populates the rowset bound to the analytic grid.

Note. You should not be adding and deleting data from the analytic grid when you are not in Freeze Column Mode; this is an unsupported feature and might cause unexpected behavior.

In addition, no layout information is available, and there is no slicer, row, or column axis.

The following is an example of populating the analytic grid with data from a normal grid in Freeze Column Mode.

```
Local Rowset &RSAGRID;
Local Rowset &RSGrid;


/* Get the rowset associated with normal grid whose primary record is */
/* QE_BAM_FACTTBL */
&RSGRID = GetLevel0()(1).GetRowset(Scroll.QE_BAM_FACTTBL);


/* Get the rowset associated with Analytic grid whose primary record is */
/* QE_BAM_CCSMOKE */
&RSAGRID = GetLevel0()(1).GetRowset(Scroll.QE_BAM_CCSMOKE);

/* Flush out existing Data from the Analytic Grid */
&RSAGRID.Flush();

/* Copy data from Normal Grid to Analytic Grid in Freeze Column Mode*/
&RSGRID.CopyTo(&RSAGRID, Record.QE_BAM_FACTTBL, Record. QE_BAM_CCSMOKE);
```

Do not use the following methods with an analytic grid that is in Freeze Column Mode:

- Analytic grid GetCubeCollection method
- Analytic grid LoadData method

- Analytic grid SetAnalyticInstance method
- Analytic grid SetLayout method
- Analytic grid SlicerVisible property

Error Handling

All the analytic type classes throw PeopleCode exceptions for any fatal error that occurs in the execution of the operation. PeopleSoft recommends enclosing your analytic model programs in try-catch statements. This way, if your program catches the exception, the message set and message number that are associated with the exception object indicate the error.

See Also

Chapter 16, "Exception Class," Try-Catch Sections, page 764

Data Types of the Analytic Grid Objects

Analytic grids are declared using the AnalyticGrid data type. For example,

```
Local AnalyticGrid &MyAnalyticGrid;
```

Analytic grid columns are declared using the AnalyticGridColumn data type. For example:

```
Local AnalyticGridColumn &MYGRIDCOL;
```

Scope of Analytic Grid Objects

Both the AnalyticGrid and AnalyticGridColumn objects can be instantiated from PeopleCode only.

An AnalyticGrid is a control on a page. You generally use these objects only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message notification, a Component Interface, and so on.

AnalyticGrid objects can be of scope local, component or global.

AnalyticGrid Class Reference

This reference for the AnalyticGrid class includes:

- AnalyticGrid class built-in function.
- AnalyticGrid class methods.

- AnalyticGrid class properties.

AnalyticGrid Class Built-in Function

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAnalyticGrid

AnalyticGrid Class Methods

In this section, we discuss the AnalyticGrid class methods. The methods are described in alphabetic order.

GetColumn

Syntax

`GetColumn``(`*`ColumnName`*`)`

Description

Use the GetColumn method to instantiate an AnalyticGridColumn object.

Note. The properties for an AnalyticGridColumn and a GridColumn are the same. Any differences are indicated in the description for the GridColumn property.

Specify the grid column name in the page field properties for that field, consisting of any combination of uppercase letters, digits and "#", "$", "@", and "_".

To specify a grid column name:

1. Open the page in Application Designer, select the analytic grid and access the Analytic Grid control properties.
2. On the General tab, type the new grid name in Page Field Name.

Parameters

Parameter	***Description***
ColumnName	Specify the name of the column that you want to access.

Returns

An AnalyticGridColumn object.

Note. The properties for an AnalyticGridColumn and a GridColumn are the same. Any differences are indicated in the description for the GridColumn property.

See Also

Chapter 20, "Grid Classes," GridColumn Class, page 1045

GetCubeCollection

Syntax

`GetCubeCollection``()`

Description

Use the GetCubeCollection method to return a reference to the cube collection associated with the analytic grid.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

Parameters

None.

Returns

A CubeCollection object.

See Also

Chapter 2, "Analytic Calculation Engine Classes," page 13

Chapter 2, "Analytic Calculation Engine Classes," CubeCollection Class, page 36

LoadData

Syntax

`LoadData``()`

Description

Use the LoadData method to cause the system to get fresh data for the grid from the analytic calculation engine. Generally, you would use this method after you perform some operation, such as SetLayout, that might change the value or layout of the data.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

Parameters

None.

Returns

None.

SetAnalyticInstance

Syntax

`SetAnalyticInstance`*`(ID)`*

Description

Use the SetAnalyticInstance method to specify the analytic instance to be associated with this analytic grid.

An AnalyticGrid object can be bound only once to an analytic instance. If the SetAnalyticInstance method is called after the analytic grid is bound to an instance, the method call has no effect.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

Parameters

Parameter	***Description***
ID	Specify the analytic instance ID as a string.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AnalyticInstance Class Methods, page 16

Chapter 2, "Analytic Calculation Engine Classes," page 13

SetLayout

Syntax

`SetLayout`*`(&SlicerArray,&RowAxisArray,&ColumnAxisArray)`*

Description

Use the SetLayout method to set the layout for the three axes, slice, row, and column.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

If you specify No Drag drop mode for an analytic grid in Application Designer, the analytic grid has the column axis, row axis and slicer axis, but the end user isn't allowed to change the layout by dragging and dropping elements between axes. However, you can still change the layout using the SetLayout method.

Note. You can specify a null value ("") for all the required parameters for this method.

Parameters

Parameter	***Description***
&SlicerArray	Specify an already instantiated array of string containing the list of fields to be put on the slice axis.
&RowAxisArray	Specify an already instantiated array of string containing the list of fields to be put on the row axis.
&ColumnAxisArray	Specify an already instantiated array of string containing the list of fields to be put on the column axis. **Note.** The maximum number of columns that can be displayed in an analytic grid is four.

Returns

None.

AnalyticGrid Class Properties

In this section we discuss the AnalyticGrid properties. The properties are described in alphabetic order.

Inactive

Description

Use this property to specify whether the analytic grid is inactive or active. This property takes a Boolean value, true if the grid is inactive, false otherwise.

If you specify this property as true, the analytic grid is not displayed to the user and no data is fetched from the database.

This property is read-write.

Label

Description

Use this property to specify the label that appears as the title of the grid.

Note. You can't use this property to set labels longer than 100 characters. If you try to set a label of more than 100 characters, the label is truncated to 100 characters. Always put any changes to labels in the Activate event for the page. This way the label is set every time the page is accessed.

This property is read-write.

ShowGridLines

Description

Use this property to specify whether grid lines are displayed with the analytic grid. This property takes a Boolean value, true to show the lines, false otherwise.

This property is read-write.

SlicerVisible

Description

Use this property to specify whether the slicer is displayed with the analytic grid. This property takes a Boolean value, true to show the slicer, false otherwise.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

This property is read-write.

SummaryText

Description

Use this property to set or return a string representing the summary text for the analytic grid.

Summary text enables you to provide a brief description of the functionality and content of the grid area. This property is pertinent for users who access the application in accessibility mode using screen readers.

This property is read-write.

Example

```
&MyAnGrid = GetAnalyticGrid(Page.PSMYPAGE, "PSMYPAGE");
&MyAnGrid.SummaryText = "This is the new summary text through PeopleCode";
```

See Also

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Creating Analytic Grids," Setting Analytic Grid Label Properties

Chapter 5

Analytic Type Classes

This chapter provides an overview of Analytic Calculation Engine classes and discuses:

- Using the analytic type classes.
- Error handling.
- Data types of the analytic type objects.
- Scope of analytic type objects.
- How to import the analytic type classes.
- How to create an analytic type class object.
- Analytic type classes reference.

Important! The analytic type classes are not supported on IBM z/OS and Linux for IBM System z platforms.

Understanding the Analytic Calculation Engine Classes

PeopleSoft Analytic Calculation Engine comprises a calculation engine plus several PeopleTools features which enable application developers to define both the calculation rules and the display of calculated data within PeopleSoft applications for the purposes of multi-dimensional reporting, data editing, and analysis.

More specifically, developers create *analytic models* in order to define the rules which are used to calculate data. Developers also create PeopleSoft Pure Internet Architecture pages with *analytic grids* in order to display the data within PeopleSoft applications. End users view, analyze and make changes to this data. When end users save their changes, PeopleSoft Analytic Calculation Engine recalculates the data and saves the calculated data to the database.

PeopleCode enables developers to manipulate analytic calculation data as follows:

- Use the Analytic Calculation Engine classes to either retrieve or specify data in an instance of an analytic model loaded into the system, and also to calculate (or recalculate) cube values.
- Use the Analytic Calculation Engine metadata classes to manipulate an analytic model definition. For example, you can add cubes to a cube collection or rename an existing user function for a model.
- Use the analytic grid classes to manipulate the display of analytic calculation data on a page.
- Use the analytic type metadata classes to manipulate an analytic type definition. For example, you can specify a new analytic model for an analytic type definition.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," page 53

Chapter 4, "Analytic Grid Classes," page 159

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine

Using the Analytic Type Classes

You can create analytic type definitions using Application Designer. PeopleSoft provides the analytic type classes for accessing these definitions at runtime.

You can also create an analytic type definition in PeopleCode, using the Create method, then saving it to the database using the Save method. After you save the definition, you can access it using Application Designer.

You only need to create an instance of the analytic type. All other objects are instantiated from the analytic type object.

Once the AnalyticTypeDefn object is created, you can then execute either the Get or Create methods to "instantiate" the analytic type object.

Error Handling

All the analytic type classes throw PeopleCode exceptions for any fatal error that occurs in the execution of the operation. PeopleSoft recommends enclosing your analytic model programs in try-catch statements. This way, if your program catches the exception, the message set and message number that are associated with the exception object indicate the error.

See Also

Chapter 16, "Exception Class," Try-Catch Sections, page 764

Data Types of the Analytic Type Objects

Every PeopleSoft analytic type object is declared as its own data type, that is, AnalyticTypeDefn objects are declared as type AnalyticTypeDefn, AnalyticTypeModelDefn objects are declared as type AnalyticTypeModelDefn, and so on.

The following are the data types for the PeopleSoft analytic type classes:

- AnalyticTypeDefn
- AnalyticTypeModelDefn

- AnalyticTypeRecordDefn

Scope of Analytic Type Objects

The analytic type objects can only be instantiated from PeopleCode.

These objects can be used anywhere you have PeopleCode, that is, in a Application Engine program, an application class, record field PeopleCode, and so on.

Analytic type objects can be of scope Local, Component or Global.

How to Import the Analytic Type Classes

The analytic type classes are *not* built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing the analytic type classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_ANALYTICTYPEDEFN contains the following subclasses:

- AnalyticTypeDefn
- AnalyticTypeModelDefn
- AnalyticTypeRecordDefn

The import statement you should use should be as follows:

```
import PT_ANALYTICTYPEDEFN:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available.

See Also

Chapter 6, "Application Classes," page 189

How to Create an Analytic Type Class Object

After you've imported the analytic type classes, you need to instantiate an instance of the AnalyticTypeDefn class, using the constructor for that class and the Create function. After you create the object, you must populate it using either the Get or Create method.

The following example creates an AnalyticTypeDefn object from the QE_ACE_ALLFUNCTION analytic type definition.

```
Local PT_ANALYTICTYPEDEFN:AnalyticTypeDefn &AnalyticType;
```

```
&AnalyticType = create PT_ ANALYTICTYPEDEFN:AnalyticTypeDefn("QE_ACE_ALLFUNCTION");
```

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Analytic Calculation Engine Metadata Classes Constructor, page 57

Analytic Type Classes Constructor

You must use the constructor for the AnalyticTypeDefn class to instantiate an instance of that class. All other objects are instantiated from this class.

AnalyticTypeDefn

Syntax

AnalyticTypeDefn*(TypeName)*

Description

Use the AnalyticTypeDefn constructor to create an AnalyticTypeDefn.

Once the AnalyticTypeDefn object is created, you can then execute either the Get or Create methods to instantiate the model.

Note. The Delete and Rename methods can only be executed before a model is instantiated.

Parameters

Parameter	***Description***
TypeName	Specify the name of an analytic type definition.

Returns

An AnalyticTypeDefn object.

Example

```
&Model = create AnalyticTypeDefn("QE_ACE_ALLFUNCTION");
```

See Also

Chapter 5, "Analytic Type Classes," AnalyticTypeDefn, page 172

AnalyticTypeDefn Class Methods

This section describes the analytic type definition class methods. The methods are discussed in alphabetical order.

AddModel

Syntax

AddModel*(ModelName,ModelType)*

Description

Use the AddModel method to add either an analytic model or optimization model to the analytic type definition. If the model doesn't exist, this method fails.

Parameters

Parameter	***Description***
ModelName	Specify the name of an existing model.
ModelType	Specify the model type. Values are:

Value	***Description***
%ModelType_ACE	The model is an Analytic Calculation Engine.
%ModelType_OPT	The model is an optimization.

Returns

The new AnalyticTypeModelDefn object.

See Also

Chapter 5, "Analytic Type Classes," DeleteModel, page 176 and Chapter 5, "Analytic Type Classes," GetModelNames, page 179

Chapter 5, "Analytic Type Classes," AnalyticTypeModelDefn Class Properties, page 183

AddRecord

Syntax

`AddRecord`*`(RecordName)`*

Description

Use the AddRecord method to add a record to the analytic type definition. If the record definition doesn't exist in Application Designer, or if the record already exists in this analytic type definition, this method fails.

Parameters

Parameter	***Description***
RecordName	Specify the name of the record that you want to add.

Returns

An AnalyticTypeRecordDefn object that represents the record.

See Also

Chapter 5, "Analytic Type Classes," DeleteRecord, page 177 and Chapter 5, "Analytic Type Classes," GetRecordNames, page 180

Chapter 5, "Analytic Type Classes," AnalyticTypeRecordDefn Class Methods, page 184

Create

Syntax

`Create``()`

Description

Use the Create method to create, and instantiate, a new analytic type. If the analytic type already exists, an exception is thrown.

Parameters

None.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," Save, page 182

Chapter 16, "Exception Class," Try-Catch Sections, page 764

CopyTo

Syntax

CopyTo*(NewAnalyticTypeName)*

Description

Use the CopyTo method to copy the current AnalyticTypeDefn object to the AnalyticTypeDefn object with the specified name. This method fails if the analytic type specified with *NewAnalyticTypeName* already exists.

Parameters

Parameter	*Description*
NewAnalyticTypeName	Specify the name of the new analytic type definition that you want to create.

Returns

The new AnalyticTypeDefn object.

See Also

Chapter 5, "Analytic Type Classes," Create, page 174 and Chapter 5, "Analytic Type Classes," Save, page 182

Delete

Syntax

`Delete()`

Description

Use the Delete method to delete the analytic type definition. You must use this method immediately after you create the AnalyticType object, before you instantiate it, that is, before you use either the Get or Create method.

Note. This method executes immediately, and deletes the definition from Application Designer.

Parameters

None.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," Create, page 174; Chapter 5, "Analytic Type Classes," Get, page 178 and Chapter 5, "Analytic Type Classes," Save, page 182

DeleteModel

Syntax

DeleteModel*(ModelName)*

Description

Use the DeleteModel method to delete the specified model from the analytic type definition.

Note. The model is not deleted from the analytic type definition until you use the Save method.

Parameters

Parameter	Description
ModelName	Specify the name of a model that exists in the analytic type definition.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," AddModel, page 173 and Chapter 5, "Analytic Type Classes," GetModelNames, page 179

Chapter 5, "Analytic Type Classes," AnalyticTypeModelDefn Class Properties, page 183

DeleteRecord

Syntax

```
DeleteRecord(RecordName)
```

Description

Use the DeleteRecord method to delete a record from the analytic type definition.

Note. The record deleted from the analytic type definition until you use the Save method.

Parameters

Parameter	Description
RecordName	Specify the name of the record that you want to remove from the analytic type definition.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," AddRecord, page 174 and Chapter 5, "Analytic Type Classes," GetRecordNames, page 180

Chapter 5, "Analytic Type Classes," AnalyticTypeRecordDefn Class Methods, page 184

Get

Syntax

`Get``()`

Description

Use the Get method to instantiate an existing analytic type definition as an AnalyticTypeDefn object. If the analytic type doesn't exist, this method throws an exception.

Parameters

None.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," Create, page 174 and Chapter 5, "Analytic Type Classes," Save, page 182

GetModel

Syntax

`GetModel``(`*`ModelName`*`)`

Description

Use the GetModel method to return a reference to an AnalyticTypeModelDefn object.

This method fails if the model specified by *ModelName* doesn't exist.

Parameters

Parameter	***Description***
ModelName	Specify the name of the analytic type model definition that you want to access.

Returns

An AnalyticTypeModelDefn object.

See Also

Chapter 5, "Analytic Type Classes," AnalyticTypeModelDefn Class Properties, page 183

GetModelNames

Syntax

GetModelNames()

Description

Use the GetModelNames method to return an array of string containing all the model names in this analytic type definition.

Parameters

None.

Returns

An array of string.

See Also

Chapter 5, "Analytic Type Classes," AddModel, page 173 and Chapter 5, "Analytic Type Classes," DeleteModel, page 176

Chapter 5, "Analytic Type Classes," AnalyticTypeModelDefn Class Properties, page 183

GetRecord

Syntax

`GetRecord``(`*`RecordName`*`)`

Description

Use the GetRecord method to return a reference to an AnalyticTypeRecordDefn object.

This method fails if the record specified by *RecordName* doesn't exist.

Parameters

Parameter	***Description***
RecordName	Specify the name of the record that you want to access.

Returns

An AnalyticTypeRecordDefn object.

See Also

Chapter 5, "Analytic Type Classes," AnalyticTypeRecordDefn Class Methods, page 184

GetRecordNames

Syntax

`GetRecordNames``()`

Description

Use the GetRecordNames method to return an array of string containing the names of all the records associated with the analytic type definition.

Parameters

None.

Returns

An array of string.

See Also

Chapter 5, "Analytic Type Classes," AddRecord, page 174 and Chapter 5, "Analytic Type Classes," DeleteRecord, page 177

Chapter 5, "Analytic Type Classes," AnalyticTypeRecordDefn Class Methods, page 184

Rename

Syntax

`Rename``(`*`NewAnalyticTypeName`*`)`

Description

Use the Rename method to rename an existing analytic type definition to a new name. You must use this method immediately after you create the AnalyticType object, before you instantiate it, that is, before you use either the Get or Create method. The new name is not saved to the database until you use the Save method.

Parameters

Parameter	***Description***
NewAnalyticTypeName	Specify the new name for the analytic type definition.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," Save, page 182

Save

Syntax

```
Save()
```

Description

Use the Save method to save any changes that were made to the AnalyticTypeDefn object to the database.

Parameters

None.

Returns

None.

AnalyticTypeDefn Class Properties

In this section we discuss the AnalyticTypeDefn class properties. The properties are described in alphabetic order.

AppClassPath

Description

Use this property to specify the full name of the application package that contains the Create, Copy and Delete methods used with this analytic type definition.

This property is read-write.

Comments

Description

Use this property to specify comments for the analytic type.

This property is read-write.

Description

Description

Use this property to specify a description for the analytic type.

This property is read-write.

Name

Description

This property specifies the name of the AnalyticTypeDefn.

This property is read-only.

OwnerID

Description

This property specifies the owner ID for this AnalyticTypeDefn.

This property is read-write.

AnalyticTypeModelDefn Class Properties

In this section we discuss the AnalyticTypeModelDefn class properties. The properties are described in alphabetic order.

Name

Description

This property specifies the name of the analytic type model.

This property is read-only.

Type

Description

This property specifies the type of the analytic type model. Values are:

Value	***Description***
%ModelType_ACE	The model is an Analytic Calculation Engine.
%ModelType_OPT	The model is an optimization.

This property is read-only.

AnalyticTypeRecordDefn Class Methods

In this section we describe the AnalyticTypeRecordDefn class methods. The methods are discussed in alphabetic order.

GetSelectedField

Syntax

`GetSelectedField``()`

Description

Use the GetSelectedField to return an array of string containing the names of the fields that are selected for this record.

Parameters

None.

Returns

An array of string.

See Also

Chapter 5, "Analytic Type Classes," SetSelectedField, page 185

SetSelectedField

Syntax

`SetSelectedFields``(`*`FieldName`*`)`

Description

Use the SetSelectedField method to specify the name of the field that is selected for this record.

Parameters

Parameter	***Description***
FieldName	Specify the name of the field that you want to select.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," GetSelectedField, page 184

UnsetSelectedField

Syntax

`UnsetSelectedField``(`*`FieldName`*`)`

Description

Use the UnsetSelectedField method to unselect the selected field.

This method throws an exception if the field specified by *FieldName* isn't selected.

Parameters

Parameter	***Description***
FieldName	Specify the name of the field that you want to deselect.

Returns

None.

AnalyticTypeRecordDefn Class Properties

In this section we discuss the AnalyticTypeRecordDefn class properties. The properties are described in alphabetic order.

All of the properties listed here can also be set for the analytic type definition using Application Designer.

See Also

Enterprise PeopleTools 8.51 PeopleBook: Analytic Calculation Engine, "Viewing and Debugging Analytic Models," Understanding Analytic Model Properties

Callback

Description

Use this property to specify whether this record has a callback. This property takes a Boolean value, true if the record has a callback, false otherwise.

This property is read-write.

Description

Description

Use this property to specify a description for the record.

This property is read-write.

Name

Description

This property specifies the name of the record.

This property is read-only.

Readable

Description

Use this property to specify whether this record is readable. This property takes a Boolean value, true if the record is readable, false otherwise.

This record is read-write.

ReadOnce

Description

Use this property to specify whether this record is specified as read once. This property takes a Boolean value, true if the record is only read once, false otherwise.

This property is read-write.

ScenarioManaged

Description

Use this property to specify if the record is specified as scenario managed. This property takes a Boolean value, true if the record is scenario managed, false otherwise.

This property is read-write.

SyncOrder

Description

This property specifies the synchronization order for the record. It returns an integer value.

This property is read-write.

Writeable

Description

Use this property to specify if a record is specified as writeable. This property takes a Boolean value, true if the record is writeable, false otherwise.

This property is read-write.

AnalyticType Classes Example

The following is a typical example of using the AnalyticType classes.

```
import PT_ANALYTICTYPEDEFN:*;

Local PT_ANALYTICTYPEDEFN:AnalyticTypeDefn &AnalyticType;

Local PT_ANALYTICTYPEDEFN:AnalyticTypeRecordDefn &AnalyticTypeRecordDefn;

&AnalyticType = create PT_ ANALYTICTYPEDEFN:AnalyticTypeDefn("BB_TEST");

&AnalyticType.Create();

&AnalyticType.AddModel("QE_BAM_GENX", %ModelType_ACE);

&AnalyticTypeRecordDefn = &AnalyticType.AddRecordl("QE_BAM_CCSMOKE");

&AnalyticTypeRecordDefn = &AnalyticType.AddRecordl("QE_BAM_FACTTBL");

&AnalyticType.Save();
```


Chapter 6

Application Classes

This chapter provides an overview of application classes and discusses the following topics:

- Application classes general structure.
- Import declarations.
- Self-reference.
- Property overrides.
- Superclass reference.
- Downcasting.
- Exception handling.
- Design of base classes.
- Declaration of application classes.
- Scope of an application class.
- Application classes built-in functions and language constructs.

Understanding Application Classes

You can create Application Packages in Application Designer. These *packages* contain application classes (and may contain other packages also). An application class, at its base level, is just a PeopleCode program. However, using the Application Packages, you can create your own classes, and extend the functionality of the existing PeopleCode classes.

The application classes provide capability beyond that offered by the existing PeopleCode classes. Unlike the existing classes, a subclass can inherit all the properties and methods of the class it extends. This provides all the advantages of true object-oriented programming:

- easier to debug because all the pieces are separate
- easier to maintain because functionality is gathered into a single place
- extensible by subclass

In addition, the application classes provide more structure. Using the Application Packages, you have a clear definition of each class, as well as its listed properties and methods. This makes it easier to create a complex program that uses many functions.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Creating Application Packages and Classes"

When Would You Use Application Classes?

Use Application classes to help structure your existing PeopleCode functions. For example, prior to Application classes, most PeopleCode functions were put into FUNCLIB records. However, in larger, more complex programs, it is sometimes difficult to find (let alone maintain) a function. You could make every function a class in a package with very little reworking of your PeopleCode (as well as making the different fields in a record a package, and combining them into a single larger package.) This provides a better visual interface for grouping your functions.

In addition, use Application classes when you have to do generic processing, and just the details are different. For example, suppose one of the processes for your application is reading a file object and doing bulk insert. The process could all be contained in a single package, with the classes and subclasses providing all the details, different kinds of reading for different types of files, different inserts based on the record, and so on.

One of the main differences between a class and a function call is that the call to the class is dynamic, like a function variable, but more closely controlled. All the calls to the class (methods) must have the same signature.

For example, suppose you want to provide a more generic sort, with comparison function at the end of it. You want to use the array class Sort method, but your process has to be generic: you aren't certain if you're comparing two records or two strings. You could define a class that had as its main method a comparison, and return a -1, 0, or 1. Then write your personalized sort, extending the array class method Sort.

Another use is for business processing. Think in terms of core functionality, with vertical product solutions that extend the basic processing by providing specifics, in other words, by extending the general classes with subclasses. This could be appropriate for sales force automation or order entry.

Generally, use application classes where you can extract the common functionality.

Note. Do not extend a SOAPDoc with an application class. This is currently not supported.

Application Classes General Structure

All application classes are contained in a package. All application classes are composed of PeopleCode. In this section, we discuss the general form of an application class, which is:

- Class name
- Class extensions
- Declaration of public external interface
- Declaration of protected instance variables and methods
- Declaration of private instance variables and methods

- Definition of methods
- Constructors

This division enables a separation of the following:

- what the class provides to other classes
- what is applicable to the entire class
- what is applicable to the definition of a method

The data types used in an application class (for methods parameters, return values, and so on) can be any PeopleCode types, including application classes. Likewise, application classes can be used anywhere in a PeopleCode program where a general data type can be used.

See Also

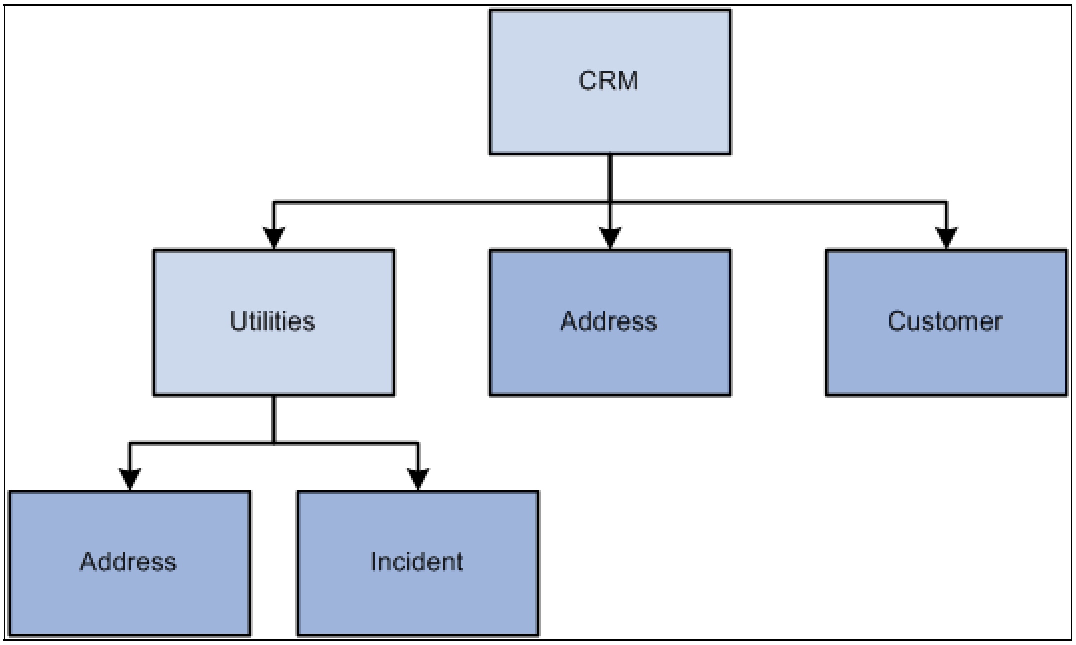
PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding the PeopleCode Language," Data Types

Class Name

Application classes have a fully qualified name that is formed hierarchically by the name of the top-level package that contains them, the package that contains that package, and so on down to the short class name, that is, the one specified when the class was created in Application Designer, using a colon for the separator between names.

The fully qualified name of the class must be unique, as this is what identifies the class. The short name of the class does not have to be unique.

For example, suppose package CRM contains the application classes Address and Customer as well as the package Utilities, which in turn contains the application classes Address and Incident:



Application classes example

There are four distinct application classes in this example:

- CRM:Address
- CRM:Customer
- CRM:Utilities:Address
- CRM:Utilities:Incident

Note. If you change the name of a package or an application class, that name change is *not* automatically propagated through all your PeopleCode programs. You must make the change manually in your programs. If you specify an incorrect name for a package or application class, you receive a warning when you try to save the PeopleCode.

Class Extension

Extension represents the "is-a" relationship. When a class extends another class, it's called a *subclass* of that class. Conversely, the class being extended is called a *superclass* of that subclass.

A subclass inherits all of the public methods and properties (collectively called *members*) of the class it extends. These members can be overridden by declarations of methods and properties in the subclass.

Note. Application classes have no multiple inheritance (that is, being able to extend more than one class.)

Type checking in PeopleCode (both at design time and runtime) does strong type checking of application classes, tracking each application class as a separate type. A subclass can be used as the class it extends, because it implements the public interfaces of its superclass. This is called *subtyping.*

In the following example, the class Banana extends the class Fruit. Banana is the *subclass* of Fruit. From Fruit, you can extend to Bananas. However, from Bananas you can't extend to Fruit. Another way to think about this is you can only extend from the general to the specific, not the other way around.

```
class Fruit
   method DoFruit();
   property number FruitNum instance;
end-class;

class Banana extends Fruit
   method DoBanana();
   property number BananaNum instance;
end-class;
```

The following code shows a correct way to assign a class object, because Banana is a *subtype* of Fruit.

```
local Fruit &Bref = Create Banana();
```

The following is *not* correct. Banana is a subtype of Fruit. Fruit is *not* a subtype of Banana. This assignment causes an error at design time.

```
local Banana &Fref = Create Fruit();
```

Before you can extend a class, you must first import it. You must import all class names you use in a PeopleCode program before you use them.

See Also

Chapter 6, "Application Classes," Import Declarations, page 209

Declaration of Public External Interface

The names of the members (that is, all the methods and properties of a class) must be unique only within that class. Different classes can have members with the same name. As members are used only with objects, and each object knows its class, the member that's accessed is the one associated with that class.

For example, more than one class has the property Name which returns the name of the object executing the property. There isn't any confusion, because the field Name property returns the name of a field, while the record name property returns the name of a record. No class has more than one property called Name.

Note. Within a class, each member name *must* be unique.

The public part of a class declaration specifies the methods and properties that the class provides to other PeopleCode programs. These methods and properties are *dynamically bound*, that is, the actual method that is called depends on the actual class of the object, as it might be an overridden method or property from a subclass.

In the following code example, the text in bold indicates the public part of the class declaration.

```
/* generic building class */
class BuildingAsset
   method Acquire ();
   method DisasterPrep();
end-class;
```

Access Control and the Declaration of Protected Properties and Methods

Application class objects have private and public access control. Properties or methods that are declared outside the "private" declaration section have public access control. This means that they can be referenced and used by any PeopleCode. This access is of course subject to access controls such as "readonly". Private properties and methods are private to the class, not instance, and can only be referenced by objects of this application class.

Between these two access control schemes of public and private, lies the concept of protected methods and properties. Protected methods and properties can be accessed only by objects of this application class and those derived from this application class. Use protected methods or properties when you want to hide them from outside use, but allow the flexibility of using them in derived classes.

The declarations of protected variables and methods are done within the class declaration, before the declaration of private variables and methods. You can use protected instance variables in interface classes. Protected methods and properties can be overridden by subclasses.

Most of the time your design can be implemented through the use of private methods and properties without resorting to the use of protected methods and properties. The following examples demonstrates the rules and some of the subtleties about when to use protected methods and properties.


```
class A;
   method A();
   property string Q;
protected
   method MP() Returns string;
   property string p;
end-class;

method A;
   &p = "Class A: property p";
end-method;

method MP
   /+ Returns String +/
   Return "Class A: method MP";
end-method;

====================

class B extends A;
   method B();
   method M(&X As C);
   method m2(&Aobj As A);
   /* property string P; */
protected
   property string Q;
   method MP() Returns string;
end-class;

method B;
   %Super = (create A());
   &Q = "Class B: property Q";
   /* &P = "Class B: property P";*/
end-method;

method M
/+ &X as FOXTEST:C +/;

   MessageBox(0, "", 0, 0, "In B:M %This.P=" | %This.p);
/* %This.p is class A property P */
   MessageBox(0, "", 0, 0, "In B:M %This.Q=" | %This.Q);
/* %This.q is class B property Q */
   MessageBox(0, "", 0, 0, %This.MP());
/* %This.MP() calls class B method MP */

   if &X.p = "Error" then
   end-if;
/* error: cannot reference &X.p since class B is
not involved in the implementation of class C */
end-method;

method m2
/+ &Aobj as FOXTEST:A +/
/* MessageBox(0, "", 0, 0, "In B:M2 &Aobj.P=" | &Aobj.p);
/* Error: cannot reference &Aobj.p from class B
since class B is not involved in its implementation */

End-method;

method MP
   /+ Returns String +/
   Return "Class B: method MP";
end-method;
```



```
=======================

class C extends A;
   method C();
protected
   property string P;
end-class;

method C;
   %Super = (create A());
   &P = "Class C: property P";
end-method;
==========================

/* AE program */
import FOXTEST:*;

Local A &A = (create A());

Local object &obj = &A;
MessageBox(0, "", 0, 0, "In AEMINITEST: &Obj.p=" | &obj.P);
/* run time error: anonymous access through type
object not allowed*/
```

The following example also illustrates some of the rules surrounding protected access control.

```
import FOXTEST:Point3d;

class Point
   method Point(&X1 As integer, &Y1 As integer);
   method Warp(&A As Point3d);
protected
   property integer x;
   property integer y;
end-class;

method Point
   /+ &X1 as Integer, +/
   /+ &Y1 as Integer +/;

   %This.x = &X1;
   %This.y = &Y1;
end-method;

method Warp
   /+ &A as FOXTEST:Point3d +/
   /* Local Integer &temp = &A.Z;  ERROR cannot access &A.Z */
end-method;
================================
import FOXTEST:Point;

class Point3d extends Point
   method Point3d(&X1 As integer, &Y1 As integer, &Z1 As integer);
   method Delta(&P As Point);
   method Delta3d(&Q As Point3d);
protected
   property integer z;
end-class;

method Point3d
   /+ &X1 as Integer, +/
   /+ &Y1 as Integer, +/
   /+ &Z1 as Integer +/;
   %Super = (create Point(&X1, &Y1));
   %This.z = &Z1;
end-method;

method Delta
   /+ &P as FOXTEST:Point +/
   /* &P.x = %This.x;
ERROR cannot access &P.x since while Point3d
(the class in which references to fields x and y occur)
is a subclass of Point (the class in which x and y are declared),
it is not involved in the implementation of Point (the type of parameter p)*/
   /* &P.y = %This.y;
ERROR cannot access &P.y. Same reason as above */
end-method;

method Delta3d
/+ &Q as FOXTEST:Point3d +/
/* The protected members of Q can be accessed because the class point3d is a⇒
 subclass of Point and is involved in the implementation of Point3d */
&Q.x = %This.x;
&Q.y = %This.y;
   &Q.z = %This.z;

end-method;
```


Declaration of Private Instance Variables and Methods

The private part of a class declaration gives the declaration of any private methods, instance variables, and private class constants. Private methods *cannot* be overridden by subclasses, because they are completely private to the declaring class. Likewise, the instance variables can't be overridden.

The declarations of private variables and methods are done within the class declaration.

Note. You cannot have private methods or instance variables in an interface type of class.

In the following example, there are both public as well as private methods and properties. Any method that is declared before the keyword Private in the initial class declaration is public. Any method declared after the keyword Private is private. Properties are only declared in Public. Instances and Constants are only declared in Private

In the following example, the class Example extends the class ExampleBase. It has both public as well as private methods and properties. It also defines the private method in the definition of methods section of the code (the private definitions are marked **like this.**)


```
import PWWPACK:ExampleBase;

class Example extends ExampleBase
   method Example();
   method NumToStr(&Num As number) Returns string;
   method AppendSlash();
   property number SlashCount get;
   property number ImportantDayOfWeek get set;
   property string SlashString readonly;
   property date ImportantDate;
private
   method NextDayOfWeek(&Dow As number) Returns date;
   Constant &Sunday = 1;
   instance string &BaseString;
end-class;

Global string &CurrentBaseString;

/* Method declarations */
method NumToStr
   return String(&num);
end-method;

method AppendSlash
   &SlashString = &SlashString | "/";
end-method;

get SlashCount
   Return Len(&SlashString) - Len(&BaseString);
end-get;

get ImportantDayOfWeek
   Return Weekday(&ImportantDate);
end-get;

set ImportantDayOfWeek
   &importantdate = %This.nextdayofweek(&newvalue);
end-set;

/* Private method. */
method nextdayofweek
   Return &ImportantDate + (&dow - Weekday(&ImportantDate));
end-method;

/* Constructor. */
method Example
   &BaseString = &CurrentBaseString;
   &SlashString = &BaseString;
   &ImportantDate = Date(19970322);
end-method;
```

Definition of Methods

After the declaration of any global or component variables and external functions needed by the methods, comes the actual definition of the methods. This section discusses some of the differences in how application programs are processed by the system, and how application class method parameters are processed.

- The system *never* skips to the next top-level statement.
- Application programs generally pass parameters by value, which is not the same as for existing functions.

- Parameter passing with object data types is by reference.
- Application programs use the out specifier to pass a parameter by reference.

Not Skipping to The Next Top-Level Statement

For existing functions and programs, the system skips to the next top-level statement in some circumstances, such as a field not found error. This never happens in an application class program. The code is always executed or an error is produced.

Passing Parameters in Application Class Methods

The parameters to a method are passed by value in the absence of an *out* specification in the parameter list. However, if a parameter list has an out specification, that argument (when used in the call of a method) must be a value that can be assigned something, and is passed by *reference*. In particular this means you cannot pass an object property to a method which requires the parameter to be an "out" parameter. Application class properties are always passed by value.

For example:

```
/* argument passed by reference */
method increment(&value as number out);


/* argument passed by value */
method increment(&value as number);
```

This is in contrast with existing PeopleCode functions that pass *all* parameters by reference when they can be assigned something. This means the method signature is a specification of which parameters can be updated by the method. This makes the code easier to debug and maintain because it's easy to see what can be updated and what can't.

The following is an example of code that increments a variable by one.

```
local Number &Mine;

&Mine = 1;

&obj.Increment(&Mine);
```

&Mine now has the value 2.

You cannot pass properties by reference. For example, supposed MyProp is a property of class Xan, a statement that requires a reference doesn't work if you supply the property. The following code will always fail because &Xan.MyProp is always passed by value and the MySqlExec method requires it to be passed by reference (so a value can be returned.)

```
MySqlExec("select from bar", &Xan.MyProp)
```

This makes sense because the semantics of an object require that you can only change a property with standard property references, not as a side affect of some other action.

Passing Parameters with Object Data Types

Parameters with object data types are always passed by reference:


```
/* argument passed by reference  */
method storeInfo(&f as File);
```

If you specify the out modifier for a method parameter with an object data type, it becomes a *reference parameter*. This means that the parameter variable is passed by reference instead of the object that it is pointing at when passed.

For example, if you pass an object parameter with the out modifier:

```
method myMethod(&arg as MyObjectClass);

Local MyObjectClass &o1 = create MyObjectClass("A");
Local MyOtherObjectClass &o2 = create MyOtherObjectClass();

&o2.myMethod(&o1);
```

And inside myMethod this occurs:

```
Method myMethod
   &arg  =  create MyObjectClass("B");
end-method;
```

Since the method argument is reassigned within the body of myMethod, &o1 does not point at the new instance of MyObjectClass (initialized with "B") after the method call completes. This is because &o1 still references the original instance of MyObjectClass.

However, if &o1 had been passed with the out modifier, after the method call completes, &o1 points at whatever the parameter was last assigned to; in this case, the new instance of MyObjectClass. The parameter, rather than the object, is passed by reference.

Using the Out Specification for a Parameter

In the following example, a class, AddStuff, has a single public method, DoAdd. This adds two numbers together, then assigns them as different numbers. In the signature of the method declaration, the first parameter is *not* declared with an out statement, while the second one is.

```
class AddStuff
   method DoAdd(&P1 as number, &P2 as number out);
end-class;

method DoAdd
   &X = &P1 + &P2;
   &P1 = 1;
   &P2 = 2;
end-method;
```

In the following PeopleCode example, an object named &Aref is instantiated from the class AddStuff. Two parameters, &I and &J are also defined.

```
local AddStuff &Aref = Create AddStuff();
local number &I = 10;
local number &J = 20;
```

The following code example is correct. &J is changed, because of the *out*statement in the method signature, and because the value is being passed by reference. The value of &I is *not* updated.

```
&Aref.DoAdd(&I, &J); /* changes &J but not &I */
```

The following code example causes a design time error. The second parameter must be passed *by reference*, not by value.

```
&Aref.DoAdd(10, 20); /* error - second argument not variable */
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding the PeopleCode Language"

Declaration of Abstract Methods and Properties

You can declare methods and properties as abstract, that is, a method or property that has a signature which specifies the parameters and results, but has no implementation in the class that defines it. The implementation may occur in one of the classes that extend the class where the method or property is initially defined.

Abstract methods and properties could be used, for example, when your application wants to provide a means for in-field customization. Your application might deliver a base class specifying the abstract methods and properties and you would allow the customization to complete those methods and properties by using a class that would extend the base class and provide implementations for some or all of the abstract methods and properties.

For a class that only contains abstract methods and properties, that is, a pure interface class, you would define a PeopleCode interface, instead of a class. You define a PeopleCode interface by using the keyword Interface instead of Class. In an interface, all the methods and properties are abstract by default.

The following example illustrates the use of abstract methods and properties. Class MyInterface is the base class which has mostly abstract methods and properties. However it is fully specified by the class MyImplementation which provides and implementation for all the methods and properties.


```
class MyInterface
   method MyMethod1() abstract;
method MyMethod2() Returns string abstract;
method MyMethod3();
   property string myproperty1 abstract readonly;
   property number myproperty2 abstract;
property number myproperty3 abstract;
end-class;
method MyMethod3
end-method;
------------------------------------------------------------------
import FOXTEST:MyInterface;

class MyImplementation extends MyInterface;
   method MyImplementation();
   method MyMethod1();
   method MyMethod2() Returns string;
   property string myproperty1 readonly;
   property number myproperty2;
   property number myproperty3;
end-class;

method MyImplementation
   %Super = Create MyInterface();[Ugh.]
end-method;

method MyMethod1
   /* dummy */
end-method;

method MyMethod2
   /+ Returns String +/
   Return "MyMethod2's implementation is this";
end-method;
```

Considerations Using Abstract Methods and Properties

Be aware of the following considerations when using abstract methods or properties.

- You cannot have private abstract methods.
- You will receive an error if you try to provide a method body for an abstract method.
- The method signatures must be identical between the abstract method definition and the method implementation in the derived subclass.
- You will receive an error if you try to provide a Get or Set body with an abstract property.
- You will receive an error and your PeopleCode program will be terminated if you try to call an abstract method or property at runtime, unless caught in a try-catch statement.
- The class that implements an interface must still assign %Super in its constructor.

Interfaces

The concept of an interface class is purely a syntactic assist when you are defining an application class which is totally composed of abstract methods and properties. In this case by simply specify the keyword **interface** you are indicating that all the methods and properties are abstract. The following two definitions are semantically equivalent.

```
class MyInterface
   method MyMethod1() abstract;
method MyMethod2() Returns string abstract;
   property string myproperty1 abstract readonly;
   property number myproperty2 abstract;
property number myproperty3 abstract;
end-class;

Interface MyInterface
   method MyMethod1();
method MyMethod2() Returns string;
   property string myproperty1 readonly;
   property number myproperty2;
property number myproperty3;
end-class;
```

When you provide an implementation for an Interface you can also use the keyword **Implements** instead of **Extends.** While this is not mandatory, specifying **Implements** describes more accurately what your application class is doing.

If your class implements an interface, you don't need to create a super object interface since the system does that automatically for you.

In addition if your constructor takes no parameters and simply exists to assign to the created instance to %super (that is, there is only one statement such as `%Super = create mySuper();` then you don't need to use a constructor at all.

Constructors

The constructor for a class is the public method with the same name as the (short name of the) class. The statements contained in this method (if any) provide the initialization of the class.

This constructor is *always* executed when an object of the class is instantiated.

Note. Before a constructor runs, the instance variables for the class are set by default based on their declared types.

Instantiate new objects of an application class by using the Create function. This function takes the name of the class, and any parameters needed for the constructor method.

If the short class name is unambiguous, that is, only one class by that short name has been imported, you can use just the short class name.

The following code instantiates an object of the class Fruit:

```
&Fref = Create Fruit();
```

If there's a possibility that the name is ambiguous, you can use the full class name to instantiate the object.

The following code instantiates an object of the Invoice class:

```
&InvObj = Create pt:examples:Invoice();
```

If the Create function is used in the context of further object expressions, that is, with a reference to members by a dot operation, the function call must be enclosed in parentheses. This is to emphasize that the creation happens before the member reference. For example:

```
&InvCust = (Create Invoice()).Cust();
```


If necessary, the constructor is supplied its parameters by the call-list after the class name in the create statement. For example:

```
&Example = create Example(&ConstructorParameter1, 2, "Third");
```

A class that does not extend some other class does not need any constructor.

A class that extends another class must have a constructor, and in the constructor, it must initialize its super class. This restriction is relaxed if the super class constructor takes no parameters, and all the constructor does is assign it to %Super. In this case, the runtime creates the super object and runs its constructor automatically. The benefit of this is two-fold: at design time you do not need to provide a constructor since the runtime automatically does the equivalent of `%Super = Create MySuperObject();` Also, at runtime, you may notice a performance increase since the super class object creation and construction are done without any PeopleCode. This latter benefit can be compounded in the case where there are multiple levels of inheritance.

To the general case, to initialize a superobject, an instance of the superclass is assigned to the keyword %Super in the constructor for the subclass. This assignment is allowed only in the constructor for the subclass.

The following example shows this type of assignment:

```
class Example extends ExampleBase
   method Example();
   ...
end-class;

Global string &CurrentBaseString;

method Example
   %Super = create ExampleBase();
   &BaseString = &CurrentBaseString;
   &SlashString = &BaseString;
   &ImportantDate = Date(19970322);
end-method
```

If your subclass' constructor only exists to create the superclass object and assign that to your %Super, you can dispense with providing a constructor completely as the following example illustrates.

```
class A
   ...
end-class;

Class B extends A
...
end-class;

Class C extends B
...
end-class;

...
Local C &c = Create C();

...
```

Classes A, B and C have no constructors and the one call to `Create C` creates objects of class C, B and A and run their constructors. PeopleSoft recommends that unless your constructors take parameters and need to do more than be assigned to their %Super, that you do not provide constructors since that simplifies design time as well as may improve runtime performance significantly.

The above example is semantically equivalent to the following, except that it may run much faster, since it does not have to run PeopleCode at each step of the construction process.

```
class A
   ...
end-class;

Class B extends A
   Method B();
...
end-class;

method B
%Super = Create A();
end-method;

Class C extends B
   Method C();
...
end-class;

method C
%Super = Create B();
end-method;
```

Note. Application classes don't have destructors, that is, a method called just before an object is destroyed. The PeopleCode runtime environment generally cleans up any held resources, so they're not as necessary as they are in other languages.

See Also

Chapter 6, "Application Classes," Superclass Reference, page 218

External Function Declarations

External function declarations are allowed in application classes, in the global and component variable declarations, after the class declaration (after the end-class statement), and before the method definitions. For example:


```
import SP:Record;

class Person extends SP:Record
   method Person(&pid As string);

   method getPerson() Returns Record;
   method getUserName() Returns string;

private
   instance Record &recPerson;

end-class;

Declare Function getUserName PeopleCode FUNCLIB_PP.STRING_FUNCTIONS FieldFormula;

method Person
   /+ &pid as String +/

   %Super = create SP:Record(Record.SPB_PERSON_TBL);
   &recPerson = %Super.getRecord();
   &recPerson.PERSON_ID.Value = &pid;

end-method;

method getPerson
   /+ Returns Record +/

   Return &recPerson;

end-method;

method getUserName
   /+ Returns String +/
   Local string &formattedName;

   &formattedName = getUserName(&recPerson.PERSON_ID);
   Return &formattedName;
end-method;
```

When application class properties and instance variables are used as the argument to functions or methods, they are always passed by value, never by reference. This means that the called function or method cannot change the value of the property or instance variable. If you want a function call to change a property or instance variable, use a local variable in the call and then assign the property or instance variable after the call.

Suppose you have the following class and you want to call Meth2 in the body of Meth1, passing it MyProp:

```
class MyClass
   property number MyProp;
   method Meth1();
   method Meth2(&i as number out);
end-class;
```

The following PeopleCode *fails* with a compile-time error:

```
method Meth1
   Meth2(%This.MyProp);   /* error - field or temp name required. */
   Meth2(&MyProp);   /* error - field or temp name required. */
end-method;
```

The following PeopleCode is valid:

```
method Meth1
/* Assignment needed if &i is input as well as output in Meth2. */
   local number &Var = %This.MyProp;
   Meth2(&Var);
   %This.MyProp = &Var;
end-method;
```

Note. You call a Java program from an Application Class the same way you do using any other PeopleCode program, that is, by using one of the existing Java class built-in functions.

See Chapter 22, "Java Class," From Java to PeopleCode, page 1090.

Naming Standards

This section describes the naming standards for:

- Packages.
- Classes.
- Methods.
- Properties.

Naming Packages

The naming standard for application packages is in the following format.

```
CC_XXX[_YYY]
```

Where CC is the company name (for example, PS is PeopleSoft), XXX is the product line, and YYY is the product code.

Examples are PS_CRM and PS_CRM_RB.

Naming Classes

Use a constructor to be the same name as the short name for the class. PeopleSoft recommends that you do not name classes to be GetXxx.

Naming Methods

Make method names simple verb or verb object words, spelled out, first letter of each word capitalized. Examples: GetField, Save, SetDefault, WriteRecord

Naming Properties

Make property names nouns, likewise spelled out, first letter of each word capitalized. Examples: Dimension, HTTPMethod, PathInfo, SubscriptionProcessId.

For Boolean read-only properties, sometimes IsXxx is appropriate. Examples: IsOpen, IsNewFieldID, IsChanged, IsDeleted

State Boolean properties positively, that is, IsChanged, as opposed to negatively, IsUnChanged. This avoids double negatives when testing (If Not &MyField.IsUnChanged Then. . . .)

Import Declarations

Before you can use a class name in a PeopleCode program (either a class definition program or an ordinary PeopleCode program), you must first import it using an import statement at the beginning of your program.

An import statement names either all the classes in a package or one particular application class.

If you're importing all the classes in a package, the syntax for the import statement ends with the following:

```
:*
```

This makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available.

If you're importing one particular application class, the syntax of the import statement ends with the short name of the class. Only that class is made available.

For example, the following imports all the classes of the package named Fruit:

```
Import Fruit:*;
```

The following imports only the application classes Banana and Kiwi. If the package named Fruit contains other classes, those are not made available to the PeopleCode program:

```
Import Fruit:Banana;

Import Fruit:Kiwi;
```

If you had a subpackage in Fruit named Drinks, you could access all the classes in that subpackage like this:

```
Import Fruit:Drinks:*;
```

A PeopleCode program (either a separate program or one contained within a class definition) can use only application class names that have been imported. However, a class definition can use its own name without having to import it.

It's possible to import two packages which contain application classes with the same short name. For example, suppose my package Fruit contained classes Apple and Banana. Suppose a second package, Drinks, contained Apple and Cherry.

```
Import Fruit:*;

Import Drinks:*;
```


You could refer to the Banana and Cherry classes using just the short name of the class. However, you couldn't refer to the Apple class by the short name. You must always use the full name reference (that is, the name of the package and the name of the class) when referring to Apple.

```
Global Banana &MyBanana;
```

```
Local Cherry &CherryTree;
```

```
Component Fruit:Apple &GreenApple;
```

The following line produces an error, because the program doesn't know which Apple class you're referring to:

```
Component Apple &GreenApple;
```

Note. Single class imports (such as, Import Fruit:Banana) are checked for existence when you save your PeopleCode.

Self-Reference

A method can refer to the current object using the %This system variable.

Due to subtyping, %This is an object of either the method's class or a subclass of the method's class.

In PeopleCode, the methods and properties of the class *cannot* be used without an object qualification.

In the following code example, the line in bold indicates an error. You can't call the class this way: you must use %This.

```
class FactorialClass
   method factorial(&I as number) returns number;
end-class;

method factorial
   if &I <= 1 then
      return 1;
   end-if;

   return &I * factorial(&I - 1); /* error - factorial undefined */

   return &I * %This.factorial(&I - 1); /* okay */
end-method;
```

One of the implications of this is that when you're writing a method, there isn't any way to guarantee you'll access your own class' version of a public method (rather then a subclass') as it might be overridden by a subclass. The only way to make sure that you can access your own method is to make it private.

If you have some action that you also want to make public, and you want guaranteed access to it, you can make the action a private method, then define a public method for it that calls the private one.

In the following example, factorial is the public method that can be used by other classes. myFactorial is the private action.


```
class FactorialClass
   method factorial(&I as number) returns number;
private
   method myFactorial(&I as number) returns number;
end-class;

method factorial
   return myFactorial(&I);
end-method;

method myFactorial
   if &I <= 1 then
      return 1;
   end-if;

   return &I * %This.myFactorial(&I - 1);

end-method;
```

If you declare an instance variable as private you can still access it as a private property in another instance of the same class. For example, given the following declaration:

```
class Example

   private

      instance number &Num;

end-class;
```

A method of Example could reference another Example instance's &Num instance variable as follows:

```
&X = &SomeOtherExample.Num;
```

Using %This with Constructors

%This when used in application class constructor code refers to the object currently being constructed. It does not refer to the dynamic this pointer of some object under construction. Given these two application classes, the PeopleCode `%This.Test()` in the Base constructor *always* refers to the Test method in the Base class.

```
class Base
   method Base();
   method Test();
   method CallTest();
   property string sTestResult;
end-class;

method Base
   %This.Test();
end-method;

method Test
   &sTestResult = "BASE_METHOD_CALLED";
end-method;

method CallTest
   %This.Test();
end-method;

import PACKAGE:Base;

class Extend extends PACKAGE:Base;
   method Extend();

   method Test();
end-class;

method Extend
   %Super = create PACKAGE:Base();
end-method;

method Test
   /+ Extends/implements PACKAGE:Base.Test +/
   %This.sTestResult = "EXTENSION_METHOD_CALLED";
end-method;
```

Even though the Extend class method provides its own Test method which overrides Base's Test method, in PeopleCode `create Extend()` which ultimately runs Base's constructor, the `%This.Test()` call in Base's constructor still references Base's Test method, not Extend's Test method, because %This always refers to the object under construction

Properties and Constants

This section discusses the following ways to use properties and constants with application classes:

- Differentiating between properties and methods.
- Overriding properties.
- Using the Get and Set keywords.
- Using read-only properties.
- Using methods and properties for collections.
- Declaring constants.

Differentiating Between Properties and Methods

A method is a procedure or routine, associated with one or more classes, that acts on an object.

A property is an attribute of an object.

PeopleSoft recommends that you use read-write or read-only properties instead of creating methods named GetXxx and SetXxx. For example, instead of creating a method GetName, that takes no parameter, just to return the name of an object, create a read-only property Name.

If your properties simply set or return the value of an instance variable, it is far more efficient to declare the property that way than have a get and set method. The overhead of having a get and set method means that each time the property is reference some PeopleCode has to be run.

Application classes can have read-only properties, but no indexed properties.

Considerations Creating Public Properties

There are two implementations of (public) properties:

- as instance variables.
- as get-set methods.

From outside the class, you cannot tell the difference between them (for example, both are dynamically bound). However, if you just want to expose an instance variable to users of the class, you can declare a property of that name and type, which declares the instance variable too:

```
class xxx
     property string StringProp;
end-class
```

If you want the property to be read-only outside the class, use the following code:

```
class xxx
     property string StringProp readonly;
end-class
```

Both of these code examples declare an instance variable called &StringProp and provide its value for getting and change its value for setting (if not readonly). You can think of this as shorthand for the usual implementation of get-set methods.

If you want to implement the same with code (perhaps deriving the property from other instance variables or other data), use the following sort of PeopleCode program:

```
class xxx
   property string StringProp get set;
end-class;

get StringProp
   return "string"; /* Get the value from somewhere. */
end-get;

set StringProp
   /* Do something with &NewValue. */
end-set;
```


To specify a read-only property, omit the set portions of the code example.

Overriding Properties

A property of a superclass can be overridden by a subclass by providing a different implementation.

The properties of a class are implemented by the keywords Get and Set, or by access to an otherwise private instance variable.

To access the property, you do not have to specify Get or Set: the context tells the processor which task you're doing. To create read-only properties, specify only a Get method, with no Set method, or for instance variable properties, specify Readonly.

For an instance variable property, there are two different ways for method code in the class itself to refer to the property:

- %This.*PropertyName*
- *&PropertyName*

The first way is dynamically bound, as it uses whatever (possibly overridden by a subclass) implementation of *PropertyName* that the current object provides.

The second way always accesses the private instance variable of the class that the method is declared in.

The following example returns strings, by converting the property. It also demonstrates overriding.


```
class A
   property number Anum;
   property string Astring get;
   property string Astring2 get;
end-class;

/* The following is a dynamic reference to the Anum property of the current⇒
 object, which might be a subclass of A, so this might not access the A⇒
 class&rsquo; implementation of Anum as an instance variable. */
   Get Astring
   return String(%This.Anum);
end-get;

/* The following is a static reference to our &Anum instance variable. Since it⇒
 does not use %This, it will not be overridden even if the current object is a⇒
 subclass of A. */
Get Astring2
   return String(&Anum);
end-get;

/* The following overrides the Anum property of our superclass,
in this case by providing another instance variable implementation.
This gives us our own &Anum instance variable. */
class B extends A
   property number Anum;
end-class;
. . .
/* Set the Anum property in B. Because of the above definitions, this sets
the &Anum instance variable in the B object, but not the &Anum instance
variable in its superobject. In the absence of other setting, the latter will
be defaulted to 0. */

local B &B = Create B();
&B.Anum = 2;

&MyValue = &B.Astring; /* &MyValue is "2" */
&MyValue = &B.Astring2; /* &MyValue is now "0" */
```

Using Get and Set with Properties

PeopleSoft recommends against using read-only properties that are public and mutable in application classes (similar to public instance variables) because this allows external code to read and change the internal state of the object without any controls. Instead, modify the property declarations with the Get and Set keywords.

There is a slight performance advantage, and you write less code, if you use the Get and Set keywords over using get and set methods.

If you define a property with the Get and Set keywords, you also must define a get or set block in the same part of the code that contains the method bodies.

Using Read-Only Properties

PeopleCode read-only properties are functionally similar to static variables, as long as the class that defines the properties initializes them and does not change them. The difference is that you need an instance of the class to access them.

For example, consider a Category object that has a CategoryType member that can have one of the following values:

- 'R' for Regular
- 'B' for Browse
- 'N' for News

You can define three read-only properties corresponding to each of these values, REGULAR, BROWSE, and NEWS, then initialize them to the appropriate values in the constructor of the class.

When you write code to evaluate the CategoryType of the Category object, perform a test like this:

```
If (&category.getCategoryType() = &constants.NEWS) then...
```

PeopleSoft recommends referring to the read-only properties, rather than their literal values, for the following reasons:

- If you spell a literal value incorrectly, the PeopleCode editor does not catch it; if you spell a read-only property wrong, it does.
- If you ever need to change the value of a read-only property, you only have to do it in one place.
- It makes the code more maintainable and reusable (because NEWS is more intelligible than 'N').

Using Methods and Properties for Collections

Application classes can contain collections. Collections of objects generally have at least the following methods and properties.

- First()
- Item*(Index)*
- Next()
- Count (as a property)

All numeric indexes in PeopleCode are one-origin, that is, start all indexes at 1, not 0.

Declaring Constants

PeopleSoft recommends that if constants are inextricably linked to a class, you should define them within that class. If the constants are more general, potentially used by multiple classes, consolidate the constants into a constants class at application-level or package-level.

Message catalog set and message numbers are good candidates to centralize into a single constants class. It can improve the readability of code and make it easier to maintain these values.

For example, you can define a constant MSG_SET_NBR, and then define constant values for each message that explain a little about what the message represents (like REQUIRED_VALUE_MISSING, UNIMPLEMENTED_METHOD, or INSUFFICIENT_PRIVILEGES).

Using Variables in Application Classes

This section discusses how to use the following variables in application classes:

- Instance variables.
- Global variables.
- Overriding variables and properties.

Placement of Variable Declarations

Variables are declared after the class definition (the end-class statement) before anything else. After you have declared a variable, you can use it anywhere in your program.

```
class Example
   method Example_M1(&par As string);
end-class;

Global boolean &b;

method Example_M1
   /+ &par as String +/
   &b = False;
end-method;
```

Declaring Private Instance Variables

A private instance variable is private to the class, not just to the object instance. For example, consider a linked-list class where one instance needs to update the pointer in another instance. Another example is the following class declaration:

```
class Example private instance number &Num; end-class;
```

A method of Example could reference another instance of the Example &Num instance variable as:

```
&X = &SomeOtherExample.Num;
```

Avoid making every instance-scoped variable a public property. You should consider each variable individually and decide if it belongs in the public interface or not. If it does, decide if the variable warrants get or set modifiers and therefore should be a public property. If the variable only serves internal purposes to the class, it should be declared as a private instance variable.

Global Variables

PeopleSoft recommends avoiding global variables in application classes unless there is a situation where no other code works. Generally, an application class should not know about anything scoped outside of its instance. There are exceptions, since not all application classes should be standalone. For example, an application class that interacts with the Component Buffer must know about the Component Buffer objects.

Overriding Variables and Properties

PeopleSoft recommends against overriding of variables and of properties.

Since instance variables are private in scope, you cannot access them in subclasses, but you can redeclare them in subclasses. The same is true of public properties; you can even explicitly access the overridden or overriding property. However, overriding is not recommended.

Superclass Reference

A method can refer to a member of its superclass by using the %Super system variable. This construction is needed only to access superclass members that are hidden by overriding members in the current class.

In the following example, the classes that extend the general BuildingAsset class each have their own method DisasterPrep, that does the specific processing, then calls the superclass (generic) version of DisasterPrep.

```
/* generic building class */
class BuildingAsset
   method Acquire();
   method DisasterPrep();
end-class;

method Acquire
   %This.DisasterPrep();
end-method;

method DisasterPrep
   PrepareForFire();
end-method;

/* building in Vancouver */
class VancouverBuilding extends BuildingAsset
   method DisasterPrep();
end-class;

method DisasterPrep
   PrepareForEarthquake();
   %Super.DisasterPrep(); /* call superclass method */
end-method;

/* building in Edmonton */
class EdmontonBuilding extends BuildingAsset
   method DisasterPrep();
end-class;

method DisasterPrep
   PrepareForFreezing();
   %Super.DisasterPrep(); /* call superclass method */
end-method;

local BuildingAsset &Building = Create VancouverBuilding();

&Building.Acquire(); /* calls PrepareForEarthquake then PrepareForFire */

&Building = Create EdmontonBuilding();

&Building.Acquire(); /* calls PrepareForFreezing then PrepareForFire */
```


Downcasting

Downcasting is the process of determining if an object is of a particular subclass type. If the object has that subtype (either the object is of that subtype, or is a subtype of it), a reference to the subobject is returned, otherwise Null is returned. In either case, the type of the resulting value is compatible with the named subclass type.

This downcast inquiry into the actual type of an object can be used when the object has been passed to some general facility, then passed back to more specific code. For example, the general facility might be a dispatch mechanism, or a new storage structure such as a binary balanced tree. These general facilities won't know about specific types, but it might be necessary for the specific program using the general facility to recover the actual type of an object that is returned to it by the general facility.

A downcast should *not* be used when you could add members to a common base class to provide the same functionality. In other words, if you find yourself using the downcast to test whether the object is one or another of several related types, you should add members to the common base class to provide the operations or properties that you are trying to access, then provide separate implementations of those actions or properties in each specific class.

If the downcast operator is to be used in the context of further object expressions (such as references to other members using dot notation) then the downcast must be enclosed in parentheses to emphasize that the downcast happens before the member reference.

```
class Fruit
   property number FruitCount;
end-class;

class Banana extends Fruit
   property number BananaCount;
end-class;
```

In the following code example, first the variables are assigned. All the variable assignments are legal, for Banana is a subtype of Fruit.

```
local Banana &MyBanana = Create Banana();
local Fruit &MyFruit = &MyBanana;  /* okay, Banana is a subtype of Fruit */
local number &Num = &MyBanana.BananaCount;
```

The next two lines of code don't produce an error, as &MyFruit is currently a Banana at runtime.

```
&MyBanana = &MyFruit as Banana;

&Num = (&MyFruit as Banana).BananaCount; /* same as &MyBanana.BananaCount */
```

In the next lines of code, we're reassigning &MyFruit to be of type Fruit, that is, of the superclass Fruit. When you try to reassign &MyFruit as Banana, &MyBanana is assigned as Null, because &MyFruit is no longer of type Banana.

```
&MyFruit = Create Fruit();

&MyBanana = &MyFruit as Banana; /* assigns Null - &MyFruit isn't a Banana */
```

Exception Handling

Use the Exception class to do exception handling in your PeopleCode. This class provides a try, catch, and throw mechanism so you don't need to check after each operation for errors. Instead, by the structure of the try-catch statements, you can declare when you are interested in handling exceptions, and how you want to handle them.

Exceptions are instances or subclasses of the PeopleCode Exception class. PeopleSoft recommends that when applicable, application class methods should throw exceptions instead of communicating back to the calling code with return values. You can create exceptions by:

- Creating an Exception base class that encapsulates the built-in function call and handles its function parameters consistently. This is the preferred way.
- Calling the built-in function CreateException.

Since a method can throw exceptions for several different reasons, PeopleSoft recommends that you:

- Create Exception subclasses often.
- Throw strongly typed exceptions.

This allows the calling code to make decisions based on the exception type.

For example, a method that retrieves customer data from a database might throw an exception named SQLException if a database error occurs, or an exception named PrivilegeException if the current user doesn't have the appropriate permissions to perform the operation. There could be different errors causing either of these exception types to be thrown in this method; if necessary, you can write the calling code to examine the specific message numbers and react accordingly.

PeopleSoft recommends that you always catch an exception; if it is not caught, it causes a runtime error. Use a try-catch block.

Not catching an exception causes a runtime error and can cause PeopleTools to display a dialog box with the message catalog entry and information about the technical context of the exception. The reasons for avoiding this by using try-catch blocks are:

- Application code often continues some form of processing even in the event of an exception; you should not assume that the script stops all processing if an exception occurs.
- The error message that PeopleTools displays might not be appropriate for end-users because of its technical information.

See Also

Chapter 16, "Exception Class," page 761

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateException

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Throw

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Try

Commenting and Documenting Application Classes

This section contains guidelines for writing method header comments, class header comments, and tags inside the comments. It discusses:

- Comments and documentation
- Method header comments
- Class header comments

Understanding Comments and Documentation

Documentation is important in all code, but even more so in application class code, which aims at, and incurs certain overhead in the name of, higher levels of reuse.

There are two types of inline comments:

- Internal comments, which start with /* and end with */.
- Comments which can potentially be used to generate API documentation, which start with /* and end with */. You can write method header and class header comments for application classes, and you can use the comment tags, with this type of comment.

Warning! In application classes you will see the use of /+ +/ style comments. *Do not use these in your PeopleCode.* These annotations are generated by the compiler. If you use them, they will be removed the next time you validate, compile, or save your PeopleCode. They are used to provide signature information on application class methods and properties, and are regenerated each time the compiler compiles your application class PeopleCode. Instead, use the standard commenting mechanisms listed above.

PeopleSoft recommends keeping a running line of asterisks along one side of an extensive comment to help the readability of the code by making the entire comment stand out from the surrounding code.

Method Header Comments

The following is an example of documentation conventions for method header comments, which are also suitable for PeopleCode. It contains comment tags, which are explained below the example.

```
/**
  * Description of what method does
  *
  * @param a optional description of param a
  * @param b optional descriptions of param b
  * @exception PrivilegeException thrown if etc, etc
  * @exception SQLException thrown if etc., etc.
  * @return String  -  description of potential values
  */
Method myMethod
   ...
End-method;
```


Method Comment Tags

The following are the suggested tags for commenting methods.

Tag	*Description*
@param *N*	Specify a description for each method parameter. The name of the parameter is specified after the keyword @param.
@exception *name*	Specify a description of each exception class. The name of the exception is specified after the keyword @exception. For example, the descriptions can explain what causes certain exceptions to be thrown.
@return *Type*	Specify a description of the return value. The type of data that is returned is specified after the keyword @return. For example, explain the potential values a method may return, whether it returns null in some circumstances, and so on.

Not all methods require documentation. For example, some simple set or get methods have method signatures that are descriptive enough. In other cases, it may suffice to include a brief sentence in the header explaining what the method does.

As with methods, PeopleCode get or set blocks should also carry method header comments.

Class Header Comments

Header comments for the class should provide descriptive information about the class, and may optionally include additional tags, indicating version and author:

```
/**
* class description
*
* @version 1.0
* @author amachado
*/
```

Class Header Comment Tags

The following are the suggested tags for commenting class headers.

Tag	*Description*
@version *number*	Specify a version number for a class.
@author *name*	Specify an author for a class.

Designing Base Classes

This section discusses some ways you can design application classes.

- Base data classes
- Abstract base classes
- Generic base classes
- Utility classes

Base Classes

Base classes contain a representation of data elements, such as documents, categories, users, and miscellaneous security-related entities.

To design the document data element to be integration-friendly, PeopleSoft recommends you start with a relatively basic base class. Then use more specialized types of documents to create a subclass from that base class, adding the necessary specialized attributes.

Even if you think of ways to use data to drive the business rules, PeopleSoft recommends that you do not place them into your data objects. Replicating business rules for every instance of a data element can be detrimental to performance.

Abstract Base Classes

Abstract base classes contain reusable logic for PeopleCode classes; you do not instantiate them directly. Instead, you instantiate classes that are subclasses of the abstract base class. Abstract base classes use placeholder methods for specialized logic, and its subclasses override those methods to provide the specialized logic.

An example of a candidate for an abstract base class is a class that is used for administering a data entity, such as an invoice. Such a class has a Save method. A Save method can use normal methods to call generic validation routines, talk to the database, and handle errors; it can use the placeholder methods for specialized logic, such as mapping a given data class to a particular database table.

PeopleSoft recommends that you program an active validation check into your placeholder method that causes a runtime error, such as throwing an UnimplementedMethodException, if an abstract base class placeholder method hasn't been overridden.

Generic Base Classes

Generic base classes contain reusable logic for PeopleCode classes; you write them to be self-contained, fully implemented, and capable of being instantiated directly. This differs from abstract base classes, which are not instantiated directly and need subclasses that override methods to provide specialized logic.

Some examples of classes that are good candidates for creating generic base classes are:

- Service classes, which typically extend a class that provides infrastructure services, such as configurable application logging.

- Data classes, which typically extend a class that enables you to plug in adapters to support sorting their instances with custom sort algorithms.

Even when a class does not seem generic because it only has a couple of subclasses, you can still define it in your design as a generic base class rather than cloning common code across different classes. The main reason is maintainability: it's easier to maintain a single class than trying to keep track of the same code scattered across your database.

Utility Classes

Some reusable code is the kind you typically don't want to inherit, but you just want to use here and there. Utility classes can serve this purpose.

String parsing, data type conversion, and formatting routines are all good candidates for utility classes. PeopleSoft recommends that anytime you code something that's more than just a line or two, and that seems to be reusable, write it as a utility class.

Declaration of Application Classes

This section discusses instructions for declaring application classes, such as:

- Declaring application classes.
- Importing class names.
- Referencing Superclasses.
- Declaring private methods.

Declaring Application Classes

Application object references are declared in your PeopleCode programs by either the short name of the class (if the short name is unambiguous amongst all the imported classes), or by the full name of the class. Like all uses of class names, to use the name in a variable declaration, you must have imported the class. Application objects can be of Local, Global, or Component scope.

Every application class is a definition of a new data type in PeopleCode, like the record, rowset, or field data types. The application class includes both the definition of its external interface, its data (properties and instance variables), and its actions (methods), just like the PeopleSoft delivered classes.

Importing Class Names

When you import a class name, fully-qualifying it is optional. The syntax is:

```
<package>:<subpackage>:<...>:<classname or wildcard>
```

PeopleSoft recommends that you individually import each class when you import multiple classes from the same package, instead of using wildcards.

Referencing Superclasses

PeopleSoft recommends that you do not set %Super to a new instance of the current class you are defining because this would start an infinite loop.

```
/* Do not do this! */
%Super = %This;
```

Declaring Private Methods

PeopleSoft recommends that you use private methods for succinct operations, even if they aren't reused, to improve the readability and maintainability of the code. You can also do this by using many inline comments.

Scope of an Application Class

Application classes can be instantiated only from PeopleCode. These classes can be used anywhere you have PeopleCode, that is, in message notifcation PeopleCode, Component Interface PeopleCode, record field PeopleCode, and so on.

Application Classes Built-in Functions and Language Constructs

In this section, we discuss the Application classes built-in functions and language constructs, in alphabetical order.

Class

Syntax

```
Class classname [{Extends | Implements} Classname]
   [Method_declarations]
   [Property_declarations]
[Protected
   [Method_declarations]
   [Instance_declarations]
   [Constant_declaration]]
[Private
   [Method_declarations]
   [Instance_declarations]
   [Constant_declaration]]
End-Class
```

Where Method_declarations are of the form:

Method*methodname*([*MethodParam1* [, *MethodParam2*. . .] [**Returns***Datatype*]] [**abstract**])

Where Property_declarations are of the form:

Property*DataTypePropertyName* {{[**Get**] | [**Set**]} | [**abstract**] | [**readonly**]}

Where Instance_declarations are of the form:

Instance*DataType&Variable1* [, *&Variable2. . .*]

Where Constant_declarations are of the form:

Constant*&Constant* = {*Number* | *String* | **True** | **False** | **Null** }

Description

Use the Class language construct to build application classes in an application package. All classes within a package must be uniquely named, however, classes contained in different packages do not have to be named uniquely.

External function declarations are allowed in application classes, in the global and component variable declarations, after the class declaration (after the end-class statement), and before the method definitions.

Parameters

Parameter	***Description***
classname	Specify the name of the class that you are creating. All classes within a package must be uniquely named, however, classes contained in different packages do not have to be named uniquely.
Extends | **Implements***classname*	For a regular class, specify the name of the class that this class extends. If the class is an interface type of class, specify the name of the interface that this class implements. You must import the class to extend it.
Method_declarations	Specify the methods (and their signature) as used in this class.
Property_declarations	Specify the properties, and their usage, in this class.
Protected	Use this keyword to declare any methods, constants, or instance variables as protected to the class, that is, are only visible to the declaring class and subclasses.
Private	Use this keyword to declare any methods, constants, or instance variables as private to the class, that is, they can't be accessed by any other classes.
Instance_declarations	Specify any variables that should be in any instance (object) of the class.
Constant_declarations	Specify any constants that should be used with this class.

Returns

None.

Example

```
class Example extends ExampleBase
   method NumToStr(&Num as number) returns string;
   method AppendSlash();
   property number SlashCount get;
   property number ImportantDayOfWeek get set;
   property string SlashString readonly;
   property date ImportantDate;
private
   method NextDayOfWeek(&DoW as number) returns date;
   constant &Sunday = 1;
   instance number &BaseString;
end-class;
```

See Also

Chapter 6, "Application Classes," Method, page 230; Chapter 6, "Application Classes," Get, page 227 and Chapter 6, "Application Classes," Set, page 231

Get

Syntax

Get*PropertyNameStatementList*
End-Get

Description

Use the Get language construct when defining properties in an application class that are implemented by methods rather than an instance variable. All properties within an application class must be uniquely named.

Parameters

Parameter	*Description*
Get *PropertyName*	Specify the name of the property that you're implementing.
StatementList	Returns the value of the property. In other words, this is a method which has no parameters and must return the value of the property.

Returns

Depends on the assignment within *StatementList*

Example

```
Get FruitCount
  Return &MyFruit.Number();
End-Get;
```

See Also

Chapter 6, "Application Classes," Method, page 230 and Chapter 6, "Application Classes," Set, page 231

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Return

Import

Syntax

Import*PackageName*:[*PackageName. . .*:]{*Classname* | *}

Description

Before you can use a class name, you must first import it using an import declaration statement at the beginning of your program.

An import declaration statement names either all the classes in a package or a single application class.

Parameters

Parameter	***Description***
Import*PackageName*	Specify the name of the package that you want to import.
*	Import all the classes of the specified package.
Classname	Import only this class of the specified package. You won't have access to any other classes in the specified package, only this one.

Returns

None.

Example

The following gives you access to all the classes in the package Fruit:

```
Import Fruit:*;
```

The following only gives you access to the class Banana in the package Fruit:

```
Import Fruit:Banana;
```

The following gives you access to the class Apple, in the package Drinks, which contains the package Fruit:

```
Import Drinks:Fruit:Apple;
```

See Also

Chapter 6, "Application Classes," Class, page 225

Interface

Syntax

```
Interface classname [{Extends  | Implements} Classname]
   [Method_declarations]
   [Property_declarations]
[Protected
   [Method_declarations]
   [Instance_declarations]
   [Constant_declaration]]
]
End-Interface
```

Where Method_declarations are of the form:

```
Methodmethodname([MethodParam1 [, MethodParam2. . .] [ReturnsDatatype]] [
abstract])
```

Where Property_declarations are of the form:

```
PropertyDataTypePropertyName {{[Get] | [Set]} | [abstract] | [readonly]}
```

Where Instance_declarations are of the form:

```
InstanceDataType&Variable1 [, &Variable2. . .]
```

Where Constant_declarations are of the form:

```
Constant&Constant = {Number  | String | True | False | Null }
```

Description

Use the Interface language construct to create classes of type interface in an application package. An interface class is a purely abstract class.

Parameters

An interface does not have Private methods or properties, but all the other parameters are identical to the Class language construct.

See Chapter 6, "Application Classes," Class, page 225.

Returns

None.

See Also

Chapter 6, "Application Classes," Class, page 225

Method

Syntax

`Method` *`MethodNameStatementList`***`End-Method`**

Description

Use the Method statement to define the methods of your application class.

Parameters

Parameter	*Description*
Method*methodname*	Specify the name of the method that you are creating.
StatementList	Specify the program of the method, what it does.

Returns

Depends on the method.

Example

```
method NumToStr
   return String(&Num);
end-method;

method AppendSlash
   &SlashString = &SlashString | "/";
end-method;
```

See Also

Chapter 6, "Application Classes," Get, page 227 and Chapter 6, "Application Classes," Set, page 231

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Return

Set

Syntax

Set*PropertyNameStatementList***End-Set**

Description

Use the Set language construct when implementing properties in an application class. All properties within an application class must be uniquely named. The new value for the property is available in the &NewValue parameter of the Set method.

Note. You cannot create a set-only property. You can create only get-set properties.

Parameters

Parameter	*Description*
Set *PropertyName*	Specify the name of the property that you're implementing.
StatementList	Change the value of the property to be &NewValue.

Returns

None.

Example

```
Set FruitCount
```

```
   &Fruitcount = &MyFruit[&NewValue].ActiveRowCount();

End-Set;
```

See Also

Chapter 6, "Application Classes," Get, page 227 and Chapter 6, "Application Classes," Method, page 230

Chapter 7

API Repository

This chapter provides an overview of the PeopleSoft API Repository and discusses:

- Repository properties.
- Bindings collection properties.
- Bindings collection methods.
- Bindings properties.
- Bindings methods.
- Namespaces collection properties.
- Namespaces collection methods.
- Namespaces properties.
- Namespaces methods.
- ClassInfo collection properties.
- ClassInfo collection methods.
- ClassInfo properties.
- MethodInfo collection properties.
- MethodInfo collection methods.
- MethodInfo properties.
- PropertyInfo collection properties.
- PropertyInfo collection methods.
- PropertyInfo properties.
- Summary of repository methods and properties.

Understanding the PeopleSoft API Repository

This section discusses:

- The PeopleSoft API Repository.

- Accessing the repository by using PeopleCode.
- Accessing the repository by using Visual Basic.

The PeopleSoft API Repository

The PeopleSoft API Repository enables PeopleCode and third-party integrators to discover the internally available classes, methods, and properties that are provided by PeopleSoft for integration. The repository is useful to third-party integrators who integrate in a generic fashion: middleware providers, testing tool providers, and automated documentation providers.

The PeopleSoft API Repository is *not*a necessary interface for integrators who integrate at the business-rule level, such as integration with an expense report, and so on. Those integrators should use component interfaces.

The repository describes available PeopleSoft APIs and provides mechanisms to determine the classes that are available in the API, the properties of each class, the methods of a class (along with the required parameters), and information concerning which group a class belongs to (known as a namespace).

The process of determining information about the API is known as *discovery.* Third-party integrators use information found through discovery to drive generic integration tools.

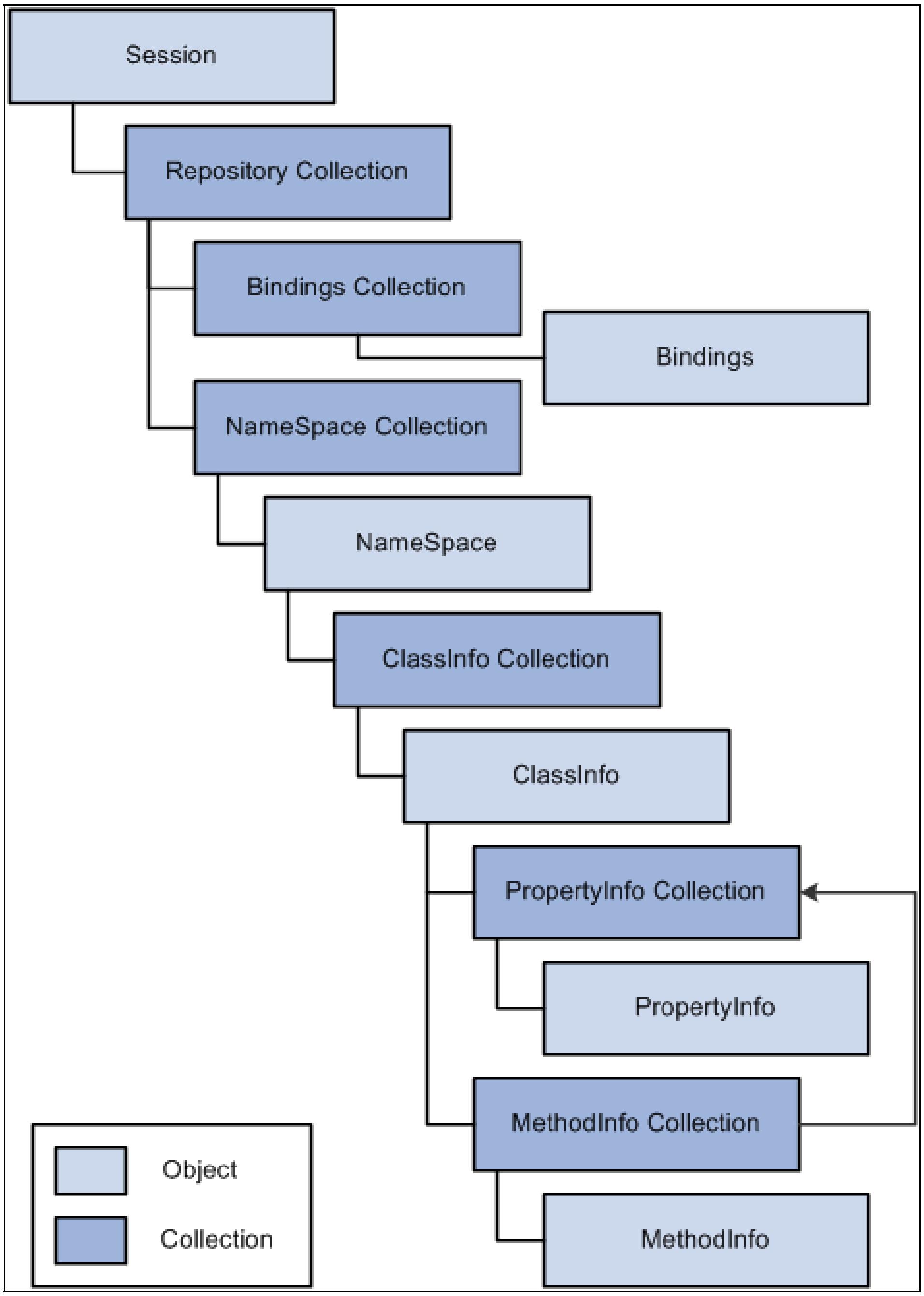
The repository is divided into namespaces. Each namespace contains a collection of related classes. Example namespaces include "PeopleSoft," "ComponentInterface," "Trees," and "BusinessInterlinks".

A *class* defines a related set of methods and properties. Using the repository, you can determine the methods and properties that are available and that can be used on any object returned by a call to the PeopleSoft API. An instance of a class is known as an object.

A *property* is a data item of an object that has both a name and type (string, number, Boolean, and so on are examples of types). Some properties are used for inputting data to a class, some are used for getting data from a class, and some are used for both. Whether a property is used for input or output or both is known as usage.

A *method* is a function that you can call on an object. Methods have a name and a return type (string, number, Boolean, and so on). Methods also have a collection of arguments that must be set prior to invoking the method. Methods arguments have identical attributes to properties.

This diagram shows the different types of objects and collections instantiated from the repository:



Repository class hierarchy

Example of Accessing the Repository Using PeopleCode

This example gets information for the class ABS_HIST from the Namespace component interface and writes it to the file BC.TXT:

This is the complete code sample, followed by the flat file. The next section presents steps that explain each line.

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;
Local string &OutTEXT;
Local File &MYFILE;

&MYSESSION = %Session;

&MYFILE = GetFile("CI.txt", "A");

&NAMESPACES = &MYSESSION.Repository.Namespaces;
&NAMESPACE = &NAMESPACES.ItemByName("CompIntfc");

&OutTEXT = "Namespace = " | &NAMESPACE.Name;
&MYFILE.WriteLine(&OutTEXT);

&CLASSES = &NAMESPACE.classes;
&CLASS = &CLASSES.ItemByName("ABS_HIST");

&OutTEXT = "  Class: " | &CLASS.Name;
&MYFILE.WriteLine(&OutTEXT);

&OutTEXT = "     Methods";
&MYFILE.WriteLine(&OutTEXT);

&METHODS = &CLASS.methods;
For &K = 0 To &METHODS.Count - 1
   &METHOD = &METHODS.item(&K);
   &OutTEXT = "             " | &METHOD.name | ":  " | &METHOD.Type

   &MYFILE.WriteLine(&OutTEXT);
   &ARGUMENTS = &METHOD.arguments;
   For &M = 0 To &ARGUMENTS.count - 1
      &ARGUMENT = &ARGUMENTS.item(&M);
      &OutTEXT = "           " | &ARGUMENT.name | ": " | &ARGUMENT.type;
      &MYFILE.WriteLine(&OutTEXT);
   End-For;
End-For;

&OutTEXT = "     Properties";
&MYFILE.WriteLine(&OutTEXT);

&PROPERTIES = &CLASS.properties;
For &I = 0 To &PROPERTIES.count - 1
   &PROPERTY = &PROPERTIES.item(&I);
   &OutTEXT = "              " | &PROPERTY.name | ": " | &PROPERTY.type;
   &MYFILE.WriteLine(&OutTEXT);
End-For;
&MYFILE.Close();
```

The previous code produces the following flat file:

```
Namespace = CompIntfc
  Class: ABS_HIST
     Methods
             Get:  Boolean
             Save:  Boolean
             Cancel:  Boolean
             Find:  ABS_HIST
             GetPropertyByName:  Variant
           Name: String
             SetPropertyByName:  Number
           Name: String
           Value: Variant
             GetPropertyInfoByName:  CompIntfcPropertyInfo
           Name: String
     Properties
              EMPLID: String
              LAST_NAME_SRCH: String
              NAME: String
              ABSENCE_HIST: ABS_HIST_ABSENCE_HISTCollection
              interactiveMode: Boolean
              getHistoryItems: Boolean
              componentName: String
              compIntfcName: String
              stopOnFirstError: Boolean
              propertyInfoCollection: CompIntfcPropertyInfoCollection
              createKeyInfoCollection: CompIntfcPropertyInfoCollection
              getKeyInfoCollection: CompIntfcPropertyInfoCollection
              findKeyInfoCollection: CompIntfcPropertyInfoCollection
```

The PeopleCode Example Explanation

This procedure goes through the PeopleCode example line by line.

To retrieve information from the API Repository:

1. Get a session object.

   Before you can access the PeopleSoft API Repository, you have to get a session object. The session controls access to PeopleSoft, provides error tracing, enables you to set the runtime environment, and so on.

   ```
   &MYSESSION = %Session;
   ```

2. Open the file.

   As this text will be written to a flat file, the next step is to open the file. If the file is already created, the new text is appended to the end of it. If the file hasn't been created, the GetFile built-in function creates the file.

   ```
   &MYFILE = GetFile("CI.txt", "A");
   ```

3. Get the namespace you want.

   Use the Namespaces property on the repository object to get a collection of available namespaces. We want to discover information about a component interface, so we specify CompIntfc in the ItemByName method to get that namespace. With ItemByName, you must specify a namespace that already exists. You'll receive a runtime error if you specify one that doesn't exist.

   ```
   &NAMESPACES = &MYSESSION.Repository.Namespaces;
   &NAMESPACE = &NAMESPACES.ItemByName("CompIntfc");
   ```


4. Write the text to the file.

   Because all of the information discovered is being written to a file, the next step is to write text to the file. This code writes the string "Namespace", followed by the name of the namespace, to the file.

```
&OutTEXT = "Namespace = " | &NAMESPACE.Name;
&MYFILE.WriteLine(&OutTEXT);
```

5. Get the class that you want and write text to the file.

   Use the Classes property on the Namespace object to get a collection of all the available classes. We want to discover information about the component interface named ABS_HIST, so we specify that using ItemByName. Then we write that information to the file.

```
&CLASSES = &NAMESPACE.classes;
&CLASS = &CLASSES.ItemByName("ABS_HIST");

&OutTEXT = "  Class: " | &CLASS.Name;
&MYFILE.WriteLine(&OutTEXT);
```

6. Get the methods and arguments, and write the information to the file.

   Use the Methods property on the Class object to get a collection of all the available methods. After you get each method and write the information to the file, loop through and find all of the arguments for the method, then write that information to the file.

```
&OutTEXT = "     Methods";
&MYFILE.WriteLine(&OutTEXT);

&METHODS = &CLASS.methods;
For &K = 0 To &METHODS.Count - 1
   &METHOD = &METHODS.item(&K);
   &OutTEXT = "             " | &METHOD.name | ":  " | &METHOD.Type;
   &MYFILE.WriteLine(&OutTEXT);
   &ARGUMENTS = &METHOD.arguments;
   For &M = 0 To &ARGUMENTS.count - 1
      &ARGUMENT = &ARGUMENTS.item(&M);
      &OutTEXT = "           " | &ARGUMENT.name | ": " | &ARGUMENT.type;
      &MYFILE.WriteLine(&OutTEXT);
   End-For;
End-For;
```

7. Get the properties and write the information to the file.

   Use the Properties property on the Class object to get a collection of all the available properties. Write each property, with its type, to the file. At the end of the program, close the file.

```
&OutTEXT = "     Properties";
&MYFILE.WriteLine(&OutTEXT);

&PROPERTIES = &CLASS.properties;
For &I = 0 To &PROPERTIES.count - 1
   &PROPERTY = &PROPERTIES.item(&I);
   &OutTEXT = "              " | &PROPERTY.name | ": " | &PROPERTY.type;
   &MYFILE.WriteLine(&OutTEXT);
End-For;
&MYFILE.Close();
```


Example of Accessing the Repository by Using Visual Basic

This example gets information for the class ABS_HIST from the Namespace component interface.

```
Private Sub Command1_Click()
    '*************************************************************
    '* TacDemo: Example Repository Usage from Visual Basic
    '*
    '* Copyright (c) 1999 PeopleSoft, Inc.  All rights reserved

    '*************************************************************

    ' Declare variables
    Dim oSession As New PeopleSoft_PeopleSoft.Session
    Dim oPSMessages As PSMessageCollection
    Dim oPSMessage As PSMessage

    ' Establish a PeopleSoft Session
    nStatus = oSession.Connect(1, "//PSOFTO070698:9001", "PTDMO", "PTDMO", 0)

    ' Enable error-handler
    On Error GoTo ErrorHandler

    ' Get a Component Interface "shell"
    Dim oNamespaces As NamespaceCollection
    Dim oNamespace As Namespace
    Dim oClasses As ClassInfoCollection
    Dim oClass As ClassInfo
    Dim oMethods As MethodInfoCollection
    Dim oMethod As MethodInfo
    Dim oArguments As PropertyInfoCollection
    Dim oArgument As PropertyInfo
    Dim oProperties As PropertyInfoCollection
    Dim oProperty As PropertyInfo

    Set oNamespaces = oSession.Repository.namespaces
    Set oNamespace = oNamespaces.ItemByName("ComponentInterface")

    Dim outText As String

    outText = "Namespace = " & oNamespace.Name & vbNewLine

    Set oClasses = oNamespace.classes
    Set oClass = oClasses.ItemByName("ABS_HIST")

    outText = outText & "   Class: " & oClass.Name & vbNewLine

    outText = outText & "       Methods" & vbNewLine

    Set oMethods = oClass.methods
    For k = 0 To oMethods.Count - 1
        Set oMethod = oMethods.Item(k)        outText = outText & "            "
& oMethod.Name & ":  " & oMethod.Type & vbNewLine
        Set oArguments = oMethod.arguments
        For m = 0 To oArguments.Count - 1
            Set oArgument = oArguments.Item(m)
            outText = outText & "                "
& oArgument.Name & ":  " & oArgument.Type & vbNewLine
        Next
    Next

    outText = outText & "       Properties" & vbNewLine

    Set oProperties = oClass.properties
    For k = 0 To oProperties.Count - 1
        Set oProperty = oProperties.Item(k)
        outText = outText & "            " & oProperty.Name & ":  "
& oProperty.Type & vbNewLine
```



```
        Next

     txtResults = outText

    ' Leave before we encounter the error handler
    Exit Sub

    ErrorHandler:
        If Err.Number = 1001 Then                ' PeopleSoft Error
            Set oPSMessages = oSession.PSMessages
            If oPSMessages.Count > 0 Then
                For i = 1 To oPSMessages.Count
                   Set oPSMessage = oPSMessages.Item(i)
                   MsgBox (oPSMessage.Text)
                Next i
                oPSMessages.DeleteAll
            Else
                MsgBox ("PS Api Error. No additional information
    available from Session log")
            End If
        Else                                    ' VB Error
            MsgBox ("VB Error: " & Err.Description)
        End If

    End Sub
```

Repository Properties

This section discusses the Repository properties in alphabetical order.

Bindings

Description

The Bindings property returns a reference to a Bindings collection.

This property is read-only.

Namespaces

Description

The Namespaces property returns a reference to a Namespaces collection.

This property is read-only.

Bindings Collection Properties

This section discusses the Bindings collection properties in alphabetical order.

Count

Description

This property returns the number of Bindings Properties objects in the Bindings collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &BINDINGS.Count;
```

See Also

Chapter 7, "API Repository," Bindings Properties, page 243

Bindings Collection Methods

This section discusses the Bindings collection methods in alphabetical order.

Item

Syntax

Item*(number)*

Description

The Item method returns a Bindings object that exists at the number position in the Bindings collection executing the method

Parameters

Parameter	*Description*
number	Specify the position number in the collection of the Bindings object that you want returned.

Returns

A reference to a Bindings object or NULL.

Example

```
For &N = 0 to &BINDINGS.Count - 1
   &BINDING = &BINDINGS.Item(&N);
   /* do processing */
End-For;
```

Bindings Properties

This section discusses the Bindings properties in alphabetical order.

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Bindings Methods

This section discusses the Bindings methods in alphabetical order.

Generate

Syntax

Generate()

Description

This method is a reserved internal function and shouldn't be used at this time.

Namespaces Collection Properties

This section discusses the Namespaces collection properties in alphabetical order.

Count

Description

This property returns the number of Namespaces Properties objects in the Namespaces collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &NameC.Count;
```

See Also

Chapter 7, "API Repository," Namespaces Properties, page 246

Namespaces Collection Methods

This section discusses the Namespaces collection methods in alphabetical order.

Item

Syntax

`Item`*`(number)`*

Description

The Item method returns a Namespaces object that exists at the *number* position in the Namespaces collection executing the method.

Parameters

Parameter	*Description*
number	Specify the position number in the collection of the Namespaces object that you want returned.

Returns

A reference to a Namespaces object or NULL.

Example

```
For &N = 0 to &NAMESPACES.Count - 1
   &NAMESPACE = &NAMESPACES.Item(&N);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName*(name)*

Description

The ItemByName method returns the item specified by *name.Name* is not case-sensitive.

Parameters

Parameter	*Description*
name	Specify the name of the Namespaces object that you want returned. This parameter takes a string value.

Returns

A reference to a Namespaces object or NULL.

Example

```
&NAMESPACE = &NAMESPACES.ItemByName("BusinessComponent");
```


Namespaces Properties

This section discusses the Namespaces properties in alphabetical order.

Classes

Description

This property returns a reference to a ClassInfo collection object.

This property is read-only.

Example

```
&CLASSC = &NAME.Classes;
```

See Also

Chapter 7, "API Repository," ClassInfo Collection Properties, page 247

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Namespaces Methods

This section discusses the Namespaces methods in alphabetical order.

CreateObject

Syntax

CreateObject*(classname)*

Description

This method is a reserved internal function and shouldn't be used at this time.

ClassInfo Collection Properties

This section discusses the ClassInfo collection properties in alphabetical order.

Count

Description

This property returns the number of ClassInfo Properties objects in the ClassInfo collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &InfoC.Count;
```

See Also

Chapter 7, "API Repository," ClassInfo Properties, page 249

ClassInfo Collection Methods

This section discusses the ClassInfo collection methods in alphabetical order.

Item

Syntax

`Item`*`(number)`*

Description

The Item method returns a ClassInfo object that exists at the *number* position in the ClassInfo collection executing the method.

Parameters

Parameter	*Description*
number	Specify the position number in the collection of the ClassInfo object that you want returned.

Returns

A reference to a ClassInfo object or NULL.

Example

```
For &N = 0 to &CLASSES.Count - 1
   &CLASS = &CLASSES.Item(&N);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName*(name)*

Description

The ItemByName method returns the item specified by *name.Name* is not case-sensitive.

Parameters

Parameter	*Description*
name	Specify the name of the ClassInfo object that you want returned. This parameter takes a string value.

Returns

A reference to a ClassInfo object or NULL.

Example

```
&CLASS = &CLASSES.ItemByName("ABS_HIST");
```


ClassInfo Properties

This section discusses the ClassInfo properties in alphabetical order.

Documentation

Description

This property doesn't actually return all the documentation for the class, just a brief description of the class as a string.

This property is read-only.

Methods

Description

This property returns a reference to a MethodInfo collection object.

This property is read-only.

See Also

Chapter 7, "API Repository," MethodInfo Collection Methods, page 250

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Properties

Description

This property returns a reference to a PropertyInfo collection object.

This property is read-only.

See Also

Chapter 7, "API Repository," PropertyInfo Collection Methods, page 253

MethodInfo Collection Properties

This section discusses the MethodInfo collection properties in alphabetical order.

Count

Description

This property returns the number of MethodInfo Properties objects in the MethodInfo collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &MethC.Count;
```

See Also

Chapter 7, "API Repository," MethodInfo Properties, page 252

MethodInfo Collection Methods

This section discusses the MethodInfo collection methods in alphabetical order.

Item

Syntax

`Item``(`*`number`*`)`

Description

The Item method returns a MethodInfo object that exists at the *number* position in the MethodInfo collection executing the method.

Parameters

Parameter	*Description*
number	Specify the position number in the collection of the MethodInfo object that you want returned.

Returns

A reference to a MethodInfo object or NULL.

Example

```
For &K = 0 To &METHODS.Count - 1
   &METHOD = &METHODS.item(&K);
   &OutTEXT = "             " | &METHOD.name | ":  " | &METHOD.Type;
   &MYFILE.WriteLine(&OutTEXT);
End-For;
```

ItemByName

Syntax

ItemByName*(name)*

Description

The ItemByName method returns the item specified by *name.Name* is not case-sensitive.

Parameters

Parameter	*Description*
name	Specify the name of the MethodInfo object that you want returned. This parameter takes a string value.

Returns

A reference to a MethodInfo object or NULL.

Example

```
&METHOD = &METHODS.ItemByName("Save");
```


MethodInfo Properties

This section discusses the MethodInfo properties in alphabetical order.

Arguments

Description

This property returns a reference to a PropertyInfo collection object.

This property is read-only.

See Also

Chapter 7, "API Repository," PropertyInfo Collection Methods, page 253

Documentation

Description

This property doesn't actually return all the documentation for the class, just a brief description of the class, as a string.

This property is read-only.

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Type

Description

This property returns the type of the method. Values include:

- Bool (Boolean).

- Number.
- Float.
- String.
- Variant.
- Blob (binary large object).
- Any API class name.

This property is read-only.

PropertyInfo Collection Properties

This section discusses the PropertyInfo collection properties in alphabetical order.

Count

Description

This property returns the number of PropertyInfo Properties objects in the PropertyInfo collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &PropC.Count;
```

See Also

Chapter 7, "API Repository," PropertyInfo Properties, page 255

PropertyInfo Collection Methods

This section discusses the PropertyInfo collection methods in alphabetical order.

Item

Syntax

`Item``(`*`number`*`)`

Description

The Item method returns a PropertyInfo object that exists at the *number* position in the PropertyInfo collection executing the method.

Parameters

Parameter	*Description*
number	Specify the position number in the collection of the PropertyInfo object that you want returned.

Returns

A reference to a PropertyInfo object or NULL.

Example

```
For &K = 0 To &PROPERTIES.Count - 1
   &PROPERTY = &PROPERTIES.item(&K);
   &OutTEXT = "             " | &PROPERTY.name | ":  " | &PROPERTY.Type;
   &MYFILE.WriteLine(&OutTEXT);
End-For;
```

ItemByName

Syntax

`ItemByName``(`*`name`*`)`

Description

The ItemByName method returns the item specified by *name.Name* is not case-sensitive.

Parameters

Parameter	*Description*
name	Specify the name of the PropertyInfo object that you want returned. This parameter takes a string value.

Returns

A reference to a PropertyInfo object or NULL.

Example

```
&PROPERTY = &PROPERTIES.ItemByName(&ldquo;GetHistoryItems&rdquo;);
```

PropertyInfo Properties

This section discusses the PropertyInfo properties in alphabetical order.

Documentation

Description

This property doesn't actually return all the documentation for the class, just a brief description of the class, as a string. This property is read-only.

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Type

Description

This property returns the data type. Values are:

- Bool (Boolean).
- Number.
- Float.
- String.
- Variant.
- Blob (binary large object).
- Any API class name.

This property is read-only.

Usage

Description

This property returns a number that describes in which direction the specified property (or argument) can be passed. The following table describes the valid values.

Value	*Description*
0	Can be passed into PeopleSoft API.
1	Can be passed out of PeopleSoft API.
2	Can be passed either into or out of PeopleSoft API.

This property is read-only.

Chapter 8

Array Class

This chapter provides an overview of arrays and discusses how to:

- Create arrays.
- Populate an array.
- Remove items from an array.
- Create empty arrays.
- Create and populate multi-dimensional arrays.
- Use flattening and promotion.
- Declare array objects.
- Understand the scope of an array object.
- Use array class built-in functions.
- Use array class methods
- Use array class properties

Understanding Arrays

An array is a collection of data storage locations, each of which holds the same type of data. Each storage location is called an element of the array. When you create an array, you don't have to declare the size of the array at declaration time. Arrays grow and shrink dynamically as you add (or remove) data. The size of an array is limited by the available memory. You can't access past the end of an array, but you can assign outside the existing boundaries, thereby growing the array.

Creating Arrays

Arrays are declared by using the Array type name, optionally followed by "of" and the type of the elements. If the element type is omitted, it is set by default to ANY.

```
Local Array of Number &MYARRAY;
Local Array &ARRAYANY;
```

Arrays can be composed of any valid PeopleCode data type, such as string, record, number, date, and so on.

PeopleSoft recommends you declare every object you use in PeopleCode. This provides some syntax checking when you save PeopleCode. It's better to find out that you misspelled the name of a method or property at design time, rather than at runtime!

Arrays can be declared as Local, Global, or Component, just like any other PeopleTools object.

Arrays can be created with one or more dimensions. An array with more than one dimension is called an array of arrays. The *dimension* of an array is the number of Array type names in the declaration. This is also called the *depth* of an array. The maximum depth of a PeopleCode array is 15 dimensions.

In the following example, &MYARRAY has three dimensions, and &MYA2 has two dimensions.

```
Local Array of Array of Array of Number &MYARRAY;
Local Array of Array &MYA2;
```

An array must always have a consistent dimension. This means that in a one-dimensional array *none* of the elements can be an array, in a two-dimensional array *all* of the elements must be one-dimensional arrays, and so on.

After you declare an array, use one of the built-in array functions to instantiate it and return an object reference to it. For example, the following creates an array containing one element of type &TEMP, whatever data type &TEMP may be.

```
&MYARRAY = CreateArray(&TEMP);
```

Or you can use the CreateArrayRept function to instantiate an array. The Rept stands for *repeat.* CreateArrayRept creates an array that contains the number of copies you specify of a particular value. The following code creates an array with three copies of the string &MYSTRING. This does *not* create a three-dimensional array, but rather creates an array that's already populated with three elements of data (`Len = 3`), each of which contain the same string (&MYSTRING).

```
&MYARRAY = CreateArrayRept(&MYSTRING, 3);
```

An array *object* can be assigned to an array *variable.* Array objects can be passed from and returned to any kind of PeopleCode function:

```
ANewFunc (&myarray);
MyFunc (&myarray);
&MyArray = YourFunc("something");
```

For example, the ReturnToServer function returns an array of nodes to which a message can be published.

Elements in an array are specified by providing a bracketed subscript after the array object reference:

```
&MyArray[1] = 123;
&temp = &memory[1][2][3];
&temp = &memory[1, 2, 3]; /* Same as preceding line.  */
MyFunc(&MyArray[7]);
MyFunc(10)[15] = "a string";
```

To access data in a two-dimensional array, you must specify both indexes. The following accesses the second item in the first subarray:

```
&VALUE  = &DOUBLE[1][2];
```

You receive an error if you use a zero or negative index in an array. Accessing an array element whose index is larger than the last array element is also an error, but storing to such an index extends the array. Any intervening elements between the former last element and the new last element are assigned a value based on the element type of the array. This is the same value as an unassigned variable of that type.

An array is an object, which means that assignments to an array are the same as for any other object. An array variable can be assigned the distinguished value NULL, which indicates the absence of any array value.

Array variables are supported for all scopes. This means that you can have local, global, and Component array variables.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReturnToServer

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

Populating an Array

There are several ways to populate an array. The example code following each of these methods creates the exact same array, with the same elements:

- Use CreateArray when you initially create the array:

```
Local Array of Number &MyArray;

&MyArray = CreateArray(100, 200, 300);
```

- Assign values to the elements of the array:

```
Local Array of Any &MYARRAY;

&MYARRAY = CreateArray();
&MYARRAY[1] = 100;
&MYARRAY[2] = 200;
&MYARRAY[3] = 300;
```

Note. Using CreateArray without any parameters creates an array of Any.

- Use the Push method to add items to the end of the array:

```
Local Array of Number &MYARRAY;
Local Number &MYNUM;

&MYARRAY = CreateArrayRept(&MYNUM, 0);
/* this creates an empty array of number */
&MYARRAY.Push(100);
&MYARRAY.Push(200);
&MYARRAY.Push(300);
```


- Use the Unshift method to add items to the beginning of the array:

```
Local Array of Number &MYARRAY;
Local Number &MYNUM;

&MYARRAY = CreateArrayRept(&MYNUM, 0);
/* this creates an empty array of number */
&MYARRAY.Unshift(300);
&MYARRAY.Unshift(200);
&MYARRAY.Unshift(100);
```

You can also use CreateArrayRept (repeat) when you initially create the array. The following example creates an array with three elements: all three elements have the same data, that is, 100:

```
Local Array of Number &MYARRAY;

&MYARRAY = CreateArrayRept(100, 3);
```

Removing Items From an Array

You can remove elements from either the start or the end of the array:

- Use the POP method to select and remove an element from the end of an array

```
Local Array of Number &MYARRAY;

&MYARRAY = CreateArray();
&MYARRAY[1] = 100;
&MYARRAY[2] = 200;
&MYARRAY[3] = 300;

&ANSWER = &MYARRAY.Pop();
```

&ANSWER will equal 300.

- Use the SHIFT method to select and remove an element from the beginning of an array

```
Local Array of Number &MYARRAY;

&MYARRAY = CreateArray();
&MYARRAY[1] = 100;
&MYARRAY[2] = 200;
&MYARRAY[3] = 300;

&ANSWER = &MYARRAY.Shift();
```

&ANSWER equals 100.

Creating Empty Arrays

If you create an array using CreateArray of any data type other than ANY, the new array is *not* empty: it contains one item. If you need to create a completely empty array that contains 0 elements, use one of the following:


```
Local Array of Number &AN;
Local Array of String &AS;
Local Array of Record &AR;
Local Array of Array of Number &AAN;
Local Record &REC;

&AN = CreateArrayRept(0,0); /* creates an empty array of number */
&AS = CreateArrayRept("", 0); /* creates an empty array of string */
&AR = CreateArrayRept(&REC, 0); /*create empty array of records */
&AAN = CreateArrayRept(&AN, 0); /*creates empty array of array of number */
&BOTH = CreateArray(CreateArrayRept("", 0), CreateArrayRept("", 0)); /* creates an⇒
 empty array of array of string */
```

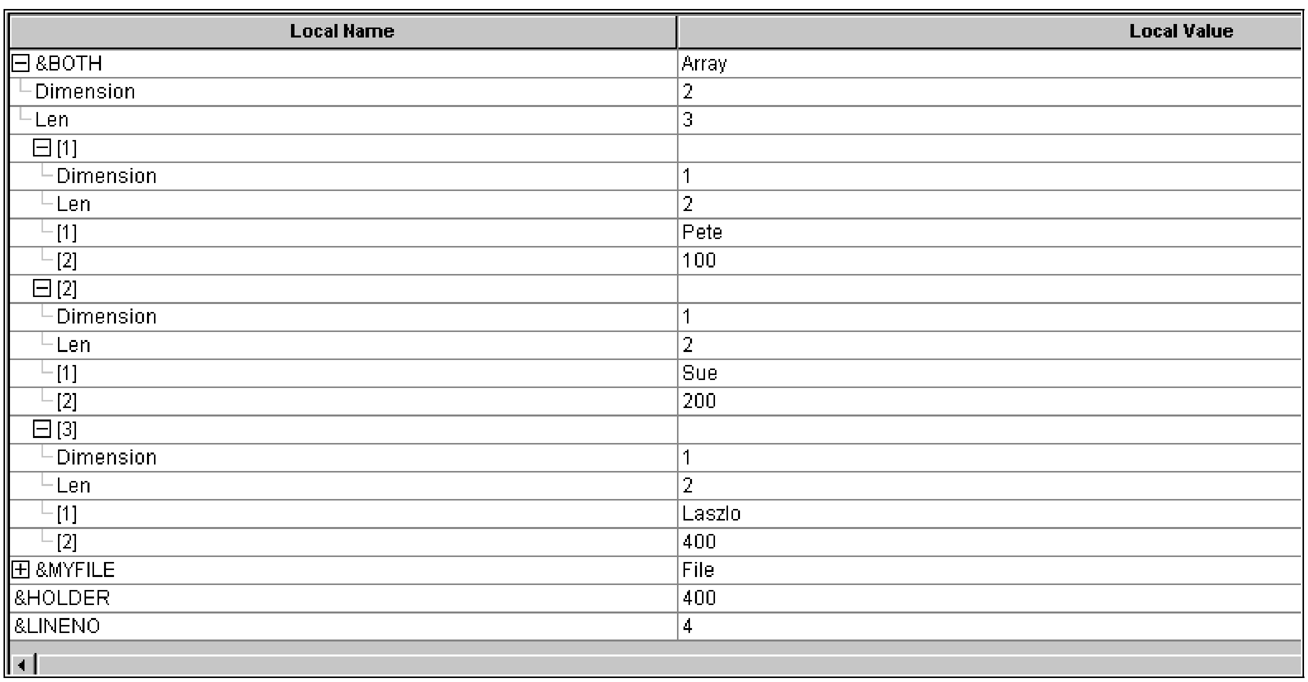
Creating and Populating Multi-Dimensional Arrays

You can create arrays with more than one dimension. Each element in a multi-dimensional array is itself an array. For example, a two-dimensional array is really an array of arrays. Each subarray has to be created and populated as an array.

Each subarray in a multi-dimensional array must be of the same type. For example, you can't create a two dimensional array that has one subarray of type string and a second subarray of type number.

The following example creates an array of array of string, then reads two files, one into each "column" of the array. The CreateArrayRept function call creates an empty array of string (that is, the Len property is 0) but with two dimensions (that is, with two subarrays, Dimension is 2). The first Push method adds elements into the first subarray, so at the end of that WHILE loop in the example, &BOTH has Len larger than 0. The other Push methods add elements to the second subarray.

```
Local array of array of string &BOTH;
Local File &MYFILE;
Local string &HOLDER;

/* Create empty &BOTH array */
&BOTH = CreateArrayRept(CreateArrayRept("", 0), 0);

/* Read first file into first column */

&MYFILE = GetFile("names.txt", "R");
While &MYFILE.ReadLine(&HOLDER);
   &BOTH.Push(&HOLDER);
End-While;

/* read second file into second column */

&MYFILE = GetFile("numbers.txt", "R");
&LINENO = 1;
While &MYFILE.ReadLine(&HOLDER);
   If &LINENO > &BOTH.Len Then
      /* more number lines than names, use a null name */
      &BOTH.Push(CreateArray("", &HOLDER));
   Else
      &BOTH[&LINENO].Push(&HOLDER);
   End-If;
   &LINENO = &LINENO + 1;
End-While;

/* if more names than numbers, add null numbers */
for &LINENO = &LINENO to &BOTH.Len
      &BOTH[&LINENO].Push("");
End-For;
```



Local Name	Local Value
&BOTH	Array
Dimension	2
Len	3
[1]	
Dimension	1
Len	2
[1]	Pete
[2]	100
[2]	
Dimension	1
Len	2
[1]	Sue
[2]	200
[3]	
Dimension	1
Len	2
[1]	Laszlo
[2]	400
&MYFILE	File
&HOLDER	400
&LINENO	4

&BOTH array expanded in PeopleCode debugger at program end

The following code reads from a two-dimensional array and writes the data from the each subarray into a separate file.


```
Local File &MYFILE1, &MYFILE2;
Local string &STRING1, &STRING2;
Local array of array of string &BOTH;
.
/* code to load data into array would be here */
.
/* open files to be written to */

&MYFILE1 = GetFile("names.txt", "A");
&MYFILE2 = GetFile("numbers.txt", "A");

/* loop through array and write to files */

For &I = 1 To &BOTH.Len
   &J = 1;
   &STRING1 = &BOTH[&I][&J];
   &MYFILE1.writeline(&STRING1);
   &J = &J + 1;
   &STRING2 = &BOTH[&I][&J];
   &MYFILE2.writeline(&STRING2);
End-For;

&MYFILE1.Close();
&MYFILE2.Close();
```

The following example populates a multi-dimensional string array using SQL. This could be used for reading small tables.

```
Component array of array of string &ArrRunStatus;

&ArrRunStatus = CreateArrayRept(CreateArrayRept("", 0), 0);
&ArrRunStatusDescr = CreateArrayRept("", 0);

&SQL = CreateSQL("SELECT FIELDVALUE, XLATSHORTNAME FROM XLATTABLE WHERE FIELDNAME ⇒
= 'RUNSTATUS'");

&LineNo = 1;
While &SQL.Fetch(&FieldValue, &XlatShortName)
      &ArrRunStatus.Push(&FieldValue);
      &ArrRunStatus[&LineNo].Push(&XlatShortName);
      &LineNo = &LineNo + 1;
End-While;
```

To search for a particular element in this array, use the following:

```
&iIndex = &ArrRunStatus.Find(&RunStatusToGet);

&RunStatusDescr = &ArrRunStatus[&iIndex][2];
```

The following example shows how to create a two-dimension array using CreateArrayRept and Push. In addition, it shows how to randomly assigns values to the cells in a two-dimension array.

```
Local array of array of string &ValueArray;

&Dim1 = 10;
&Dim2 = 5;
&ValueArray = CreateArrayRept(CreateArrayRept("", 0), 0);
For &I = 1 To &Dim1
   &ValueArray.Push(CreateArrayRept("", &Dim2));
End-For;

&ValueArray[1][1] = "V11";
&ValueArray[2][1] = "V21";

WinMessage("&ValueArray[1][1] = " | &ValueArray[1][1] | " " | "&ValueArray[2][1] =⇒
 " | &ValueArray[2][1], 0);
```

Using Flattening and Promotion

Several of the functions and methods that support arrays in PeopleCode use flattening and promotion to convert their operands to the correct dimension for the array.

Flattening converts an array into its elements. For example, the CreateArray built-in function constructs an array from its parameters. If it is constructing a one-dimensional array and is given an array as a parameter, then it flattens that array into its elements and adds each of them to the array that it is building, rather than adding a reference to the array (which would be a dimension error) or reporting an error.

Likewise, for functions that operate on multiple-dimension arrays, if they are given a non-array parameter, they use *promotion* to convert it into an array of suitable dimension. For example, the Push method appends elements onto the end of an array. If it is operating with a two-dimensional array of Array of Number, and is given a numeric argument, it will convert the argument into a one-dimensional array of Number with the given number as its only element, and then append that to the two-dimensional array.

An array value can only be assigned to an array variable if the value and variable have both the same dimension and base type. This means you cannot assign an Array of Any to an Array of Number variable or vice-versa. You can, however, assign an Array of Number to an Any variable, as long as you do not break the rule that the base element of an array cannot be an array reference value.

Declaring Array Objects

Arrays are declared by using the Array type name, optionally followed by "of" and the type of the elements. If the element type is omitted, it is set by default to ANY.

```
Local Array of Number &MYARRAY;
Local Array &ARRAYANY;
```

Arrays can be composed of any valid PeopleCode data type, such as string, record, number, date, and so on.

Understanding the Scope of an Array Object

An array object can only be instantiated from PeopleCode. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

Array Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateArray

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateArrayAny

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateArrayRept

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Split

Array Class Methods

In this section, we discuss the array class methods.

Clone

Syntax

```
Clone()
```

Description

The Clone method returns a reference to a new array, which is a copy of the given array. It copies all levels of the array, meaning, if the array contains elements which are themselves arrays (actually references to arrays), then the copy contains references to copies of the subarrays. Furthermore, if the array contains elements that are references to the same subarray, then the copy contains references to *different* subarrays (which of course have the same value).

Parameters

None. The array object that the clone method is executed against is the array to be cloned. Assigning the result of this method assigns a reference to the new array.

Returns

An array object copied from the original.

Example

In the following example, &AAN2 contains the three elements like &AAN, but they are distinct arrays. The last line changes only &AAN2[1][1], *not* &AAN[1][1].

```
Local Array of Array of String &AAN, &AAN2;

&AAN = CreateArray(CreateArray("A", "B"), CreateArray("C", "D"), "E");
&AAN2 = &AAN.Clone();
&AAN2[1][1] = "Z";
```

After the following example, &AAN contains three elements: two references to the subarray that was &AAN[2] (with elements C and D), and a reference to a subarray with element E.

```
&AAN[1] = &AAN[2];
```

After the following example, &AAN2 contains three elements: references to two different subarrays both with elements C and D, and a subarray with element E.

```
&AAN2 = &AAN.Clone();
```

See Also

Chapter 8, "Array Class," Subarray, page 282

Find

Syntax

Find*(value)*

Description

For a one-dimensional array, the Find method returns the lowest index of an element in the array that is equal to the given value. If the value is not found in the array, it returns zero.

For a two-dimensional array, the Find method returns the lowest index of a subarray which has its first element equal to the given value. If such a subarray is not found in the array, it returns zero.

Note. This method works with arrays that have only one or two dimensions. You receive a runtime error if you try to use this method with an array that has more than two dimensions.

Parameters

Parameter	Description
value	The string or subarray to search for.

Returns

An index of an element or zero.

Example

Given an array &AS containing (A, B, C, D, E), the following code sets &IND to the index of D, that is, &IND has the value 4:

```
Local array of string &AS;

&AS = CreateArrayRept("", 0);
&AS.Push("A");
&AS.Push("B");
&AS.Push("C");
&AS.Push("D");
&AS.Push("E");
&IND = &AS.Find("D");
```

Given an array of array of string &AABYNAME containing (("John", "July"), ("Jane", "June"), ("Norm", "November"), the following code sets &IND to the index of the subarray starting with "Jane", that is, &IND has the value 2:

```
&NAME = "Jane";
&IND = &AABYNAME.Find(&NAME);
```

See Also

Chapter 8, "Array Class," Replace, page 275 and Chapter 8, "Array Class," Substitute, page 282

Get

Syntax

Get*(index)*

Description

Use the Get method to return the *index* element of an array. This method is used with the Java PeopleCode functions, instead of using subscripts (which aren't available in Java.)

Using this method is the same as using a subscript to return an item of an array. In the following example, the two lines of code are identical:

```
&Value = &MyArray[8];
```

```
&value = &MyArray.Get(8);
```

Parameters

Parameter	***Description***
index	The array element to be accessed.

Returns

An element in an array.

See Also

Chapter 8, "Array Class," Set, page 278

Chapter 22, "Java Class," page 1085

Join

Syntax

`Join``([`*`separator`*` [, `*`arraystart,arrayend`*`  [,`*`stringsizehint`*`]]])`

Description

The Join method *converts* the array that is executing the method into a string by converting each element into a string and *joining* these strings together, separated by *separator.*

Note. Join does *not* join two arrays together.

Each array or subarray to be joined is preceded by the string given by *arraystart* and followed by the string given by *arrayend.* If the given array is multi-dimensional, then (logically) each subarray is first joined, then the resulting strings are joined together.

Parameters

Parameter	*Description*
separator	Specifies what the elements in the resulting string should be separated with in the resulting string. *Separator* is set by default to a comma (",").
arraystart	Specifies what each array or subarray to be joined should be preceded with in the resulting string. *arraystart* is set by default to a left parenthesis ("(").
arrayend	Specifies what each array or subarray to be joined should be followed by in the resulting string. a*rrayend* is set by default to a right parenthesis (")").
stringsizehint	Specify a hint to the Join method about the resulting size of the string. This can improve performance if your application is concatenating a large number of string. See the Example section below.

Returns

A string containing the converted elements of the array.

Example

The following example:

```
Local array of array of number &AAN;

&AAN = CreateArray(CreateArray(1, 2), CreateArray(3, 4), 5);
&STR = &AAN.Join(", ");
```

produces in &STR the string:

```
((1, 2), (3, 4), 5)
```

The following example makes use of the *stringsizehint* parameter. The following application class passes the resulting string size hint in the Value property.

```
class StringBuffer
   method StringBuffer(&InitialValue As string, &MaxSize As integer);
   method Append(&New As string);
   method Reset();
   property string Value get set;
   property integer Length readonly;
private
   instance array of string &Pieces;
   instance integer &MaxLength;
end-class;

method StringBuffer
   /+ &InitialValue as String, +/
   /+ &MaxSize as Integer +/
   &Pieces = CreateArray(&InitialValue);
   &MaxLength = &MaxSize;
   &Length = 0;
end-method;

method Reset
   &Pieces = CreateArrayRept("", 0);
   &Length = 0;
end-method;

method Append
   /+ &New as String +/
   Local integer &TempLength = &Length + Len(&New);
   If &Length > &MaxLength Then
      throw CreateException(0, 0, "Maximum size of StringBuffer exceeded(" | &Max⇒
Length | ")");
   End-If;
   &Length = &TempLength;
   &Pieces.Push(&New);
end-method;

get Value
   /+ Returns String +/
   Local string &Temp = &Pieces.Join("", "", "", &Length);
   /* collapse array now */
   &Pieces = CreateArrayRept("", 0);
   &Pieces.Push(&Temp); /* start out with this combo string */
   Return &Temp;
end-get;

set Value
   /+ &NewValue as String +/
   /* Ditch our current value */
   %This.Reset();
   &Pieces.Push(&NewValue);
end-set;
```

The following code concatenates strings.

```
While &file.ReadLine(&line)
   &S.Append(&line);
   &S.Append(&separator);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Split

Next

Syntax

Next(&*index*)

Description

The Next method increments the given index variable. It returns true if and only if the resulting index variable refers to an existing element of the array. Next is typically used in the condition of a WHILE clause to process a series of array elements up to the end of the array.

&index must be a variable of type integer, or of type Any initialized to an integer, as Next attempts to update it.

If you want to start from the first element of the array, start Next with an index variable with the value zero. The first thing Next does is to increment the value by one.

Parameters

Parameter	*Description*
&index	The array element where processing should start. *&index* must be a variable of type integer, or of type Any initialized to an integer, as Next attempts to update it.

Returns

True if the resulting index refers to an existing element of the array, False otherwise.

Example

Next can be used in a While loop to iterate through an array in the following manner:

```
&INDEX = 0;
While &A.Next(&INDEX)
   /* Process &A[&INDEX] */
End-While;
```

In the following code example, &BOTH is a two-dimensional array. This example writes the data from each subarray in &BOTH into a different file.

```
&I = 0;
While &BOTH.Next(&I)
   &J = 1;
   &STRING1 = &BOTH[&I][&J];
   &MYFILE1.writeline(&STRING1);
   &J = &J + 1;
   &STRING2 = &BOTH[&I][&J];
   &MYFILE2.writeline(&STRING2);
End-While;
```

See Also

Chapter 8, "Array Class," Len, page 285

Pop

Syntax

Pop()

Description

The Pop method removes the last element from the array and returns its value.

Parameters

None.

Returns

The value of the last element of the array. If the last element is a subarray, the subarray is returned.

Example

The Pop method can be used with the Push method to use an array as a stack. To put values on the end of the array, use Push. To take the values back off the end of the array, use Pop.

Suppose we have a two-dimensional array &SUBPARTS which gives the subparts of each part of some assemblies. Each row (subarray) of &SUBPARTS starts with the name of the part, and then has the names of the subparts. Assuming there are no "loops" in this data, the following code puts all the subparts, subsubparts, and so on, of the part given by &PNAME into the array &ALLSUBPARTS, in "depth first order" (that is, subpart1, subparts of subpart1, …, subpart2, subparts of subpart2, …). We stack the indexes into &SUBPARTS when we want to go down to the subsubparts of the current subpart.


```
Local array of array of string &SUBPARTS;
Local array of string &ALLSUBPARTS;
Local array of array of number &STACK;
Local array of number &CUR;
Local string &SUBNAME;

/* Set the ALLSUBPARTS array to an empty array of string.  */

&ALLSUBPARTS = CreateArrayRept("dummy", 0);

/* Start with the part name.  */

&STACK = CreateArray(CreateArray(&SUBPARTS.Find(&PNAME), 2));
While &STACK.Len > 0
   &CUR = &STACK.Pop();
   If &CUR[1] <> 0 And
         &CUR[2] <= &SUBPARTS[&CUR[1]].Len Then

      /* There is a subpart here.  Add it.  */
      &SUBNAME = &SUBPARTS[&CUR[1], &CUR[2]];
      &ALLSUBPARTS.Push(&SUBNAME);

      /* Tour its fellow subparts later.  */
      &STACK.Push(CreateArray(&CUR[1], &CUR[2] + 1));

      /* Now tour its subsubparts.  */
      &STACK.Push(CreateArray(&SUBPARTS.Find(&SUBNAME), 2));

   End-If;
End-While;
```

See Also

Chapter 8, "Array Class," Push, page 273; Chapter 8, "Array Class," Replace, page 275; Chapter 8, "Array Class," Shift, page 278 and Chapter 8, "Array Class," Unshift, page 284

Push

Syntax

Push*(paramlist)*

Where *paramlist* is an arbitrary-length list of values in the form:

value1 [, *value2*] ...

Description

The Push method adds the values in *paramlist* onto the end of the array executing the method. If a value is not the correct dimension, it is flattened or promoted to the correct dimension first, then the resulting values are added to the end of the array.

Considerations Using Arrays With Object References

This method only adds an element to the end of an array. It does not clone or otherwise deep-copy the parameters. For example, if you are adding a reference to an object, Push just adds a reference to the object at the end of the array. This is similar to an assignment. It is *not* making a copy of the object. The following code snippet only puts a reference to the same record onto the end of the array.

```
While &SQL.Fetch(&Rec);
&MYARRAY.Push(&Rec);
...
End-While;
```

Even though the array is growing, all the elements point to the same record. You have only as many standalone record objects as you create. The following code snippet creates new standalone records, so each element in the array points to a new object:

```
local Record &FetchedRec = CreateRecord(Record.PERSONAL_DATA);

While &SQL.Fetch(&FetchedRec)
     &MYARRAY.Push(&FetchedRec);
     &FetchedRec = CreateRecord(Record.PERSONAL_DATA);
End-While;
```

Parameters

Parameter	***Description***
paramlist	An arbitrary-length list of values, separated by commas.

Returns

None.

Example

The following example loads an array with data from a database table.

```
Local array of record &MYARRAY;
Local SQL &SQL;

&I = 1;
&SQL = CreateSQL("Select(:1) from %Table(:1) where EMPLID like &lsquo;8%&rsquo;",⇒
 &REC);
While &SQL.Fetch(&REC);
   &MYARRAY.Push(CreateRecord(RECORD.PERSONAL_DATA));
   &I = &I + 1;
   &REC.CopyFieldsTo(&MYARRAY[&I]);
End-While;
```

See Also

Chapter 8, "Array Class," Pop, page 272; Chapter 8, "Array Class," Replace, page 275; Chapter 8, "Array Class," Shift, page 278 and Chapter 8, "Array Class," Unshift, page 284

Chapter 8, "Array Class," Using Flattening and Promotion, page 264

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

Replace

Syntax

`Replace``(`*`start, length`*`[, `*`paramlist`*`])`

Where *paramlist* is an arbitrary-length list of values in the form:

`value1`` [, `*`value2`*`] ...`

Description

Replace replaces the *length* elements starting at *start* with the given values, if any. If *length* is zero, the insertion takes place *before* the element indicated by *start.* Otherwise, the replacement starts with *start* and continues up to and including *length,* replacing the existing values with *paramlist.*

If a negative number is used for *start,* it indicates the starting position relative to the last element in the array, such that −1 indicates the position just *after* the end of the array. To insert at the end of the array (equivalent to the Push method), use a *start* of −1 and a *length* of 0.

If a negative number is used for *length,* it indicates a length measuring downward to lower indexes. Both flattening and promotion can be applied to change the dimension of the supplied parameters to match the elements of the given array.

Similar to how the built-in function Replace is used to update a string, the Replace method is a general way to update an array, and can cause the array to grow or shrink.

See Chapter 8, "Array Class," Using Flattening and Promotion, page 264.

Using Replace to Remove an Element

You can use the Replace method to remove an element from an array. Just specify the item you want replaces, with *length* equal to one.

The following example removes the item from &Index:

```
&Array.Replace(&Index, 1);
```


Parameters

Parameter	*Description*
start	Specifies where to start replacing the given elements in the array. If a negative number is used for *start,* it indicates the starting position relative to the last element in the array.
length	Specifies the number of elements in the array to be replaced.
paramlist	Specifies values to be used to replace existing values in the array. This parameter is optional.

Returns

None.

Example

For example, given the following array:

```
Local array of string &AS;
```

```
&AS = CreateArray("AA", "BB", "CC");
```

After executing the next code, the array &AN will contain four elements, ZZ, YY, BB, CC:

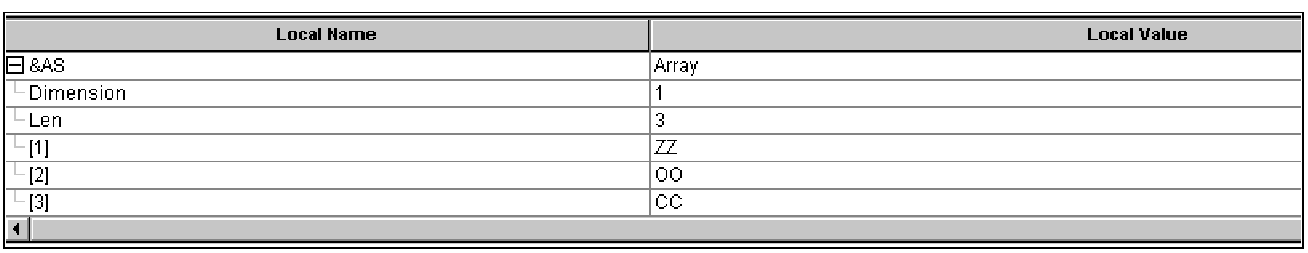
```
&AS.Replace(1, 1, "ZZ", "YY");
```

After executing the next code, the array &AN will contain three elements, ZZ, MM, CC:

```
&AS.Replace(2, 2, "MM");
```

After executing the next code, the array &AN will contain three elements, ZZ, OO, CC.

```
&AS.Replace( - 2, - 1, "OO");
```



&AS expanded in PeopleCode debugger

See Also

Chapter 8, "Array Class," Substitute, page 282 and Chapter 8, "Array Class," Find, page 266

Reverse

Syntax

```
Reverse()
```

Description

The Reverse method reverses the order of the elements in the array.

If the array is composed of subarrays, the Reverse method reverses only the elements in the super-array, it doesn't reverse all the elements in the subarrays. For example, the following:

```
&AN = CreateArray(CreateArray(1, 2), CreateArray(3, 4), CreateArray(5, 6)).reverse⇒
();
```

results in &AN containing:

```
 ((5,6), (3,4), (1,2)
```

Parameters

None.

Returns

None.

Example

Suppose you had the following array.

```
Local Array of Sting &AS;
```

```
&AS = CreateArray("R", "O", "S", "E");
```

If you executed the Reverse method on this array, the elements would be ESOR.

See Also

Chapter 8, "Array Class," Sort, page 279

Set

Syntax

`Set``(`*`index`*`)`

Description

Use the Set method to set the value of the *index* element of an array. This method is used with the Java PeopleCode functions, instead of using subscripts (which aren't available in Java.)

Using this method is the same as using a subscript to reference an item of an array. In the following example, the two lines of code are identical:

```
&MyArray[8] = &MyValue;
```

```
&MyArray.Set(8) = &MyValue;
```

Parameters

Parameter	***Description***
index	The array element to be accessed.

Returns

None.

See Also

Chapter 8, "Array Class," Get, page 267

Chapter 22, "Java Class," page 1085

Shift

Syntax

`Shift``()`

Description

Use the Shift method to remove the first element from the array and return it. Any following elements are "shifted" to an index of one less than they had.

Parameters

None.

Returns

Returns the value of the first element of the array. If the first element is a subarray, the subarray is returned.

Example

```
For &I = 1 to &ARRAY.Len;
   &ITEM = &ARRAY.Shift;
   /* do processing */
End-For;
```

See Also

Chapter 8, "Array Class," Pop, page 272; Chapter 8, "Array Class," Push, page 273; Chapter 8, "Array Class," Replace, page 275 and Chapter 8, "Array Class," Unshift, page 284

Sort

Syntax

Sort*(order)*

Description

The Sort method rearranges the elements of the array executing the method into an order.

The type of sort done by this function, that is, whether it is a linguistic or binary sort, is determined by the Sort Order Option on the PeopleTools Options page.

If the array is one-dimensional, the elements are arranged in either ascending or descending order.

The type of the first element is used to determine the kind of comparison to be made. Any attempt to sort an array whose elements are not all of the same type results in an error.

If *order* is "A", the order is ascending; if it is "D", the order is descending. The comparison between elements is the same one as if done using the PeopleCode comparison operators (<, >, =, and so on.)

Note. If you execute this method on a server, the string sorting order is determined by the character set and localization of the server.

If the array is two-dimensional, the subarrays are arranged in order by the first element of each subarray. Sorting an array whose subarrays have different types of first elements will result in an error. The comparison is done by using the PeopleCode comparison operators (<, >, =, and so on.)

Note. This method works with arrays that have only one or two dimensions. You receive a runtime error if you try to use this method with an array that has more than two dimensions.

Parameters

Parameter	***Description***
order	Specifies whether the array should be sorted in ascending or descending order. Values for *order* are:

Value	***Description***
A	Ascending
D	Descending

Returns

None.

Example

The following example changes the order of the elements in array &A to be ("Frank", "Harry", "John").

```
&A = CreateArray("John", "Frank", "Harry");
```

```
&A.Sort();
```

The following example changes the order of the elements in array &A to be (("Frank", 1957), ("Harry", 1928), ("John", 1952)).

```
&A = CreateArray(CreateArray("John", 1952), CreateArray("Frank", 1957), Create⇒
Array("Harry", 1928));
```


Local Name	Local Value
&A	Array
Dimension	2
Len	3
[1]	
Dimension	1
Len	2
[1]	John
[2]	1952
[2]	
Dimension	1
Len	2
[1]	Frank
[2]	1957
[3]	
Dimension	1
Len	2
[1]	Harry
[2]	1928

&A expanded in PeopleCode debugger

```
&A.Sort("A");
```

Local Name	Local Value
&A	Array
Dimension	2
Len	3
[1]	
Dimension	1
Len	2
[1]	Frank
[2]	1957
[2]	
Dimension	1
Len	2
[1]	Harry
[2]	1928
[3]	
Dimension	1
Len	2
[1]	John
[2]	1952

&A expanded in PeopleCode debugger, showing code results

See Also

Chapter 8, "Array Class," Reverse, page 277

PeopleTools 8.51 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," PeopleTools Options

Subarray

Syntax

Subarray*(start, length)*

Description

The Subarray method creates a new array from an existing one, taking the elements from *start* for a total of *length.* If *length* if omitted, all elements from *start* to the end of the array are used.

If the array is multi-dimensional, the subarrays of the created array are *references* to the same subarrays from the existing array. This means if you make changes to the original subarrays, the referenced subarrays are also changed. To make distinct subarrays, use the Clone method.

Parameters

Parameter	***Description***
start	Specifies where in the array to begin the subarray.
length	Specifies the number of elements in the array to be part of the subarray.

Returns

An array object.

Example

To make a distinct array from a multi-dimensional array, use the following:

```
&A = &AAN.Subarray(1, 2).Clone();
```

See Also

Chapter 8, "Array Class," Clone, page 265

Substitute

Syntax

Substitute*(old_val, new_val)*

Description

The Substitute method replaces every occurrence of a value found in an array with a new value. To replace an element that occurs in a specific location in an array, use Replace.

If the array is one-dimensional, Substitute replaces every occurrence of the *old_val* in the array with *new_val.*

If the array is two-dimensional, Substitute replaces every subarray whose first element is equal to *old_val,* with the subarray given by *new_val.*

Note. This method works with arrays that have only one or two dimensions. You receive a runtime error if you try to use this method with an array that has more than two dimensions.

Parameters

Parameter	***Description***
old_val	Specifies the existing value in the array to be replaced.
new_val	Specifies the value with which to replace occurrences of *old_val.*

Returns

None

Example

The following example changes the array &A to be ("John", "Jane", "Hamilton").

```
&A = CreateArray();
&A[1] = "John";
&A[2] = "Jane";
&A[3] = "Henry";
&A.Substitute("Henry", "Hamilton");
```

The following example changes the array &A to be (("John", 1952), ("Jane", 1957), ("Hamilton", 1971), ("Frank", 1961)).

```
&A = CreateArray(CreateArray("John", 1952), CreateArray("Jane", 1957), CreateArray⇒
("Henry", 1928), CreateArray("Frank", 1961));

&A.Substitute("Henry", CreateArray("Hamilton", 1971));
```

See Also

Chapter 8, "Array Class," Find, page 266 and Chapter 8, "Array Class," Replace, page 275

Unshift

Syntax

Unshift*(paramlist)*

Where *paramlist* is an arbitrary-length list of values in the form:

value1 [*, value2*] ...

Description

The Unshift method adds the given elements to the *start* of the array. Any following elements are moved up to indexes that are larger by the number of values moved. Flattening and Promotion are used to change the dimension of the supplied parameters to be one less than that of the given array.

Parameters

Parameter	*Description*
paramlist	Specifies values to be added to the start of the array.

Returns

None.

Example

The following code changes &A to be ("x", "Y", "a", "B", "c").

```
&A = CreateArray("a", "B", "c");
```

```
&A.Unshift("x", "Y");
```

See Also

Chapter 8, "Array Class," Pop, page 272; Chapter 8, "Array Class," Push, page 273; Chapter 8, "Array Class," Replace, page 275; Chapter 8, "Array Class," Shift, page 278 and Chapter 8, "Array Class," Using Flattening and Promotion, page 264

Array Class Properties

In this section, we discuss the array class properties.

Dimension

Description

The Dimension property is the number of "Array" type names from the declaration of the array, also called subarrays. This property returns a number.

This property is read-only.

Example

The following example sets &DIM to 2.

```
Local Array of Array of Number &AAN;

&DIM = &AAN.Dimension;
```

Len

Description

The Len property is the current number of elements in the array. This property can be updated. Setting it to a negative value results in an error.

If this property is set to a smaller (nonnegative) number than its current value, the array is truncated to that length, discarding any elements whose indexes are larger than the given new length.

If this property is set to a number larger than its current value, the array is extended to the new length. Any new elements are set to a default value based on the element type of the array.

This property is read-write.

Example

The following is a test of whether an array is empty:

```
If &ARR.Len = 0 then

/* &ARR is empty.  */

End-If;
```

Chapter 9

BPEL Classes

This chapter discusses the BPEL classes, as well as describes:

- Scope of the BPEL classes
- Data types of the BPEL classes
- How to import the BPEL classes
- How to create a BPEL object
- BPEL reference material

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Integrating with BPEL Process-Based Services"

Understanding the BPEL Classes

Use the Business Process Execution Language (BPEL) classes for launching, as well as controlling, a BPEL process instance. A BPEL process instance is associated with a service operation, that is, a message transaction, such as an outbound asynchronous message. You must have first created the service operation using PeopleSoft Pure Internet Architecture pages. When you define the service operation you specify the node used for connecting to the third-party system, as well as the routings used by the messages, and so on.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Integrating with BPEL Process-Based Services"

Scope of the BPEL Classes

The BPEL classes can be instantiated only from PeopleCode.

The BPEL classes can be called from a component, an internet script, Integration Broker, or an Application Engine program.

BPEL classes can be of Local, Global, or Component scope.

Data Types of the BPEL Classes

Every BPEL class is its own data type. That is, the class of BPELUtil is declared as type BPELUtil, and so on.

The following are the data types of the BPEL classes:

- AsyncFFSend
- BPELUtil
- IBUtil

How to Import the BPEL Classes

The BPEL classes are *not* built-in classes, like rowset, field, record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing the BPEL classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_BPEL contains the BPEL classes that you will use:

- AsyncFFSend
- BPELUtil
- IBUtil

The import statement you should use is as follows:

```
import PT_BPEL:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available.

See Also

Chapter 6, "Application Classes," page 189

How to Create a BPEL Object

After you've imported the BPEL classes, you instantiate an object of one of those classes using the constructor for the class and the Create function.

The following example creates a new instance of the BPELUtil class, as the variable &MyBU, with local scope:

```
import PT_BPEL:*;

Local BPELUtil &MyBU = Create BPELUtil();
```

See Also

Chapter 9, "BPEL Classes," BPEL Classes Constructors, page 289

BPEL Classes Constructors

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the BPEL classes.

AsyncFFSend

Syntax

AsyncFFSend()

Description

Use the AsyncFFSend constructor to instantiate an AsyncFFSend object.

Parameters

None.

Returns

A reference to an AsyncFFSend object.

See Also

Chapter 9, "BPEL Classes," AsyncFFSend Class, page 291

BPELUtil

Syntax

BPELUtil()

Description

Use the BPELUtil constructor to instantiate a BPELUtil object.

Parameters

None.

Returns

A reference to a BPELUtil object.

See Also

Chapter 9, "BPEL Classes," BPELUtil Class, page 292

IBUtil

Syntax

IBUtil()

Description

Use the IBUtil constructor to instantiate an IBUtil object.

Parameters

None.

Returns

A reference to an IBUtil object.

See Also

Chapter 9, "BPEL Classes," IBUtil Class, page 298

AsyncFFSend Class

This class provides the application handler used when an asynchronous one way operation is called from the local system. This class extends the integration broker ISend class OnRequestSend method.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

AsyncFFSend Class Method

This section describes the AsyncFFSend class method OnRequestSend.

OnRequestSend

Syntax

`OnRequestSend`*`(&Msg)`*

Description

Use the OnRequestSend method to implement the integration broker ISend class OnRequestSend method.

This method is generally used to associate the WSA_MessageID when making an asynchronous one way request to the BPEL process so that it can be tracked.

Parameters

Parameter	*Description*
&Msg	Specify an already populated and instantiated message object.

Returns

A message object.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Understanding Sending and Receiving Messages

BPELUtil Class

This class provides the utility methods to interact with BPEL processes.

BPELUtil Class Methods

This section discusses the BPELUtil class methods, in alphabetical order.

GetASyncBPELProcessInstanceUrl

Syntax

`GetASyncBPELProcessInstanceUrl``(`*`TransactionID`*`)`

Description

Use the GetASyncBPELProcessInstanceUrl method to return the complete URL that can be used to monitor an asynchronous BPEL process instance. The URL can be used to access the BPEL monitor for this particular BPEL process instance.

Parameters

Parameter	***Description***
TransactionID	Specify the transaction ID for the asynchronous BPEL process instance that you want to monitor.

Returns

A string containing a complete URL if successful, a Null or invalid string otherwise.

See Also

Chapter 9, "BPEL Classes," GetSyncBPELProcessInstanceUrl, page 294

GetBPELProcessBrowserUrl

Syntax

`GetBPELProcessBrowserUrl`*`(NodeName)`*

Description

Use the GetBPELProcessBrowserUrl to return a complete URL of the WSIL process browser. It can be used to discover all the web services that are being hosted by the given BPEL domain.

Parameters

Parameter	***Description***
NodeName	Specify the name of the node associated with the BPEL process that you want to access, as a string.

Returns

A string containing the complete URL if successful, an invalid URL string otherwise.

GetOperationType

Syntax

`GetOperationType`*`(OperationName)`*

Description

Use the GetOperationType method to return the type of the specified operation.

Parameters

Parameter	***Description***
OperationName	Specify the name of the operation for which you want to determine the type, as a string.

Returns

A string containing the operation type. Valid values are:

Value	*Description*
S	The operation type is synchronous.
A	The operation type is asynchronous.
Null	There is an error in the method execution.

GetSyncBPELProcessInstanceUrl

Syntax

`GetSyncBPELProcessInstanceUrl(NodeName,MessageID)`

Description

Use the GetSyncBPELProcessInstanceUrl method to return the complete URL that can be used to monitor the specified synchronous BPEL process instance.

Parameters

Parameter	*Description*
NodeName	Specify the node name associated with the synchronous BPEL process instance, as a string.
MessageID	Specify the message ID associated with the external system for this BPEL process instance, as a string.

Returns

A string containing a complete URL if successful, an invalid URL otherwise.

See Also

Chapter 9, "BPEL Classes," GetASyncBPELProcessInstanceUrl, page 292

LaunchASyncBPELProcess

Syntax

LaunchASyncBPELProcess*(operationName,&Msg,Username,Password)*

Description

Use the LaunchASyncBPELProcess method to invoke an asynchronous BPEL process.

Parameters

Parameter	***Description***
OperationName	Specify the name of the operation associated with the asynchronous BPEL process that you want to start, as a string.
&Msg	Specify an already instantiated and populated message object that is created from the message definition that is associated with the asynchronous BPEL process.
Username	*Username* and *Password* are used to override the security credentials that are sent by default by Integration Broker when invoking a secure BPEL process (a web service.) If you specify an empty string, the Integration Broker credentials are used. If you specify a value, it's sent as part of the security credentials of the BPEL process.
Password	*Username* and *Password* are used to override the security credentials that are sent by default by Integration Broker when invoking a secure BPEL process (a web service.) If you specify an empty string, the Integration Broker credentials are used. If you specify a value, it's sent as part of the security credentials of the BPEL process.

Returns

A string that contains the transaction ID of the published message if successful, a Null string otherwise.

See Also

Chapter 9, "BPEL Classes," LaunchSyncBPELProcess, page 296

LaunchSyncBPELProcess

Syntax

`LaunchSyncBPELProcess``(operationName,&Msg,Username,Password)`

Description

Use the LaunchSyncBPELProcess method to invoke a synchronous BPEL process.

Parameters

Parameter	***Description***
OperationName	Specify the name of the operation associated with the synchronous BPEL process that you want to start, as a string.
&Msg	Specify an already instantiated and populated message object that is created from the message definition that is associated with the synchronous BPEL process.
Username	*Username* and *Password* are used to override the security credentials that are sent by default by Integration Broker when invoking a secure BPEL process (a web service.) If you specify an empty string, the Integration Broker credentials are used. If you specify a value, it's sent as part of the security credentials of the BPEL process.
Password	*Username* and *Password* are used to override the security credentials that are sent by default by Integration Broker when invoking a secure BPEL process (a web service.) If you specify an empty string, the Integration Broker credentials are used. If you specify a value, it's sent as part of the security credentials of the BPEL process.

Returns

A message object populated with the reply message if successful, a Null object otherwise.

See Also

Chapter 9, "BPEL Classes," LaunchASyncBPELProcess, page 295

UpdateConnectorResponseProperties

Syntax

`UpdateConnectorResponseProperties``(&Msg)`

Description

Prior to sending the response message, use this method to update the connector properties for an asynchronous PeopleSoft request/response web service that is consumed by a BPEL process. Among several connector property changes that it makes, this method adds a "content-type" connector property of "text/xml."

Note. The UpdateConnectorResponseProperties method is not required with Oracle BPEL Process Manager 10.1.2 or earlier supported versions of Oracle BPEL Process Manager; however, it can be used with those versions.

Parameters

Parameter	*Description*
&Msg	The message object to be sent to the BPEL process as the response of an asynchronous PeopleSoft request/response web service.

Returns

None

Example

The following sample handler PeopleCode program processes requests for an asynchronous PeopleSoft request/response web service. A call is made to UpdateConnectorResponseProperties just before sending the response.


```
import PS_PT:Integration:INotificationHandler;
import PT_BPEL:BPELUtil;

class InboundASyncResponseHandler implements PS_PT:Integration:INotificationHandler
   method OnNotify(&MSG As Message);
end-class;

method OnNotify
   /+ &MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/
   Local PT_BPEL:BPELUtil &bpel;
   Local Message &response;
   Local string &payload;
   Local XmlDoc &xml;
   Local File &MYFILE;
   &bpel = create PT_BPEL:BPELUtil();
   &payload = "<?xml version='1.0'?><PSFTCalcResponseMessage  xmlns='http:⇒
//xmlns.oracle.com/Enterprise/Tools/schemas/PSFTCALCRESPONSEMESSAGE.V1'><result⇒
 xmlns=''>9</result></PSFTCalcResponseMessage>";
   &xml = CreateXmlDoc(&payload);
   &response = CreateMessage(Operation.PSFTASYNCCALCULATE, %IntBroker_Response);
   &response.SetXmlDoc(&xml);
   &response.IBInfo.WSA_MessageID = &MSG.IBInfo.WSA_MessageID;
   &response.IBInfo.WSA_ReplyTo = &MSG.IBInfo.WSA_ReplyTo;
   &bpel.UpdateConnectorResponseProperties(&response);
   %IntBroker.Publish(&response);
end-method;
```

IBUtil Class

The IBUtil class provides the utility methods to access integration broker metadata.

IBUtil Class Methods

This section discusses the BPELUtil class methods, in alphabetical order.

GetBPELConsoleUrl

Syntax

GetBPELConsoleUrl*(NodeName)*

Description

Use the GetBPELConsoleUrl method to return the BPEL console URL.

A transaction ID is an internal identifier for the message that is published. A message ID is the external ID that is used by the external system for identifying the transaction. The transaction ID is appended to the URL generated by this method. This concatenated URL is used to monitor the BPEL process instance.

Use the GetMessageId method to return the message ID.

Parameters

Parameter	***Description***
NodeName	Specify the name of the node associated with the BPEL process that you want to monitor.

Returns

A string containing a complete URL if successful, Null otherwise.

See Also

Chapter 9, "BPEL Classes," GetMessageId, page 300

GetBPELDomain

Syntax

GetBPELDomain*(Nodename)*

Description

Use the GetBPELDomain method to determine the name of the domain of the specified node. This is used to construct the URL for accessing a BPEL process instance.

Parameters

Parameter	***Description***
Nodename	Specify the name of the node that you want to determine the domain for, as a string.

Returns

A string containing the domain if successful, Null otherwise.

GetMessageId

Syntax

`GetMessageId`_`(TransactionId)`_

Description

Use the GetmessageId method to return the message ID used to monitor a BPEL process instance.

A transaction ID is an internal identifier for the message that is published. A message ID is the external ID that is used by the external system for identifying the transaction.

Parameters

Parameter	***Description***
TransactionId	Specify the transaction ID associated with the BPEL process instance that you want to monitor, as a string.

Returns

A string containing the message ID, Null otherwise.

See Also

Chapter 9, "BPEL Classes," GetBPELConsoleUrl, page 298

GetNode

Syntax

`GetNode`_`(MessageId)`_

Description

Use the GetNode method to return the node name of the node involved in processing of the message.

Parameters

Parameter	*Description*
MessageId	Specify the message ID associated with the BPEL process instance that you want to get the node for, as a string.

Returns

A string containing the node name if successful, Null otherwise.

GetNumberOfRoutings

Syntax

`GetNumberOfRoutings``(`*`ServiceOperationName`*`)`

Description

Use the GetNumberOfRoutings method to return the number of routings for the specified service operation.

Parameters

Parameter	*Description*
ServiceOperationName	Specify the name of the service operation that you want to discover the number of routings for, as a string.

Returns

A number.

GetStatus

Syntax

`GetStatus``(`*`TransactionId`*`)`

Description

Use the GetStatus method to return the status of the Integration Broker process specified by the transaction Id.

Parameters

Parameter	*Description*
TransactionId	Specify the transaction ID for the BPEL process instance that you want to check the status of, as a string.

Returns

A string containing the status, Null otherwise.

GetTransactionId

Syntax

`GetTransactionId``(`*`MessageID`*`)`

Description

Use the GetTransactionId method to return the transaction ID from the message. You must use the Publish or SyncRequest method on the message before you use this method.

Parameters

Parameter	*Description*
MessageID	Specify the message ID associated with the BPEL process instance, as a string.

Returns

A string containing the transaction ID associated with the published message, Null otherwise.

Chapter 10

Business Interlink Class

This chapter provides an overview of Business Interlink class and discusses the following topics:

- Using the Interlink object
- Deciding which methods to use
- Using the incoming Business Interlink methods and properties
- Understanding the state of an Interlink object
- Using Business Interlink with Application Engine
- Using the Data Type of a Business Interlink object
- Understanding the scope of a Business Interlink object
- Business Interlink class built-in function
- Business Interlink class reference
- Configuration parameters

Note. PeopleSoft Business Interlinks is a deprecated product. The Business Interlinks class currently exists for backward compatibility only. For new integrations, use Integration Broker instead.

See Also

Chapter 25, "Message Classes," page 1245

Understanding Business Interlink Class

The PeopleSoft Business Interlink framework provides a gateway for PeopleSoft applications to the services of any external system. This framework enables any PeopleSoft component (that is, a page, an Application Engine program, and so on) to integrate with any external system in near real-time and batch modes.

This framework enables a PeopleCode program to map the input and outputs of a Business Interlink definition to PeopleCode variables and record fields, then, using an Interlink object, call the Interlink plug-in, which in turn calls the external system, passes the information to the external system, and returns values to the PeopleCode program.

This documentation describes only the Interlink object portion of the PeopleSoft Business Interlink framework.

Each Interlink object is based on a Business Interlink definition, which is created in Application Designer. An Interlink object can be based on only a single Business Interlink definition.

See Also

See additional Business Interlink documentation available on PeopleSoft Customer Connection.

Using the Interlink Object

After you instantiate an Interlink object, use the Interlink object to:

- Populate the input buffer
- Send the data to the Interlink plug-in (by using the Execute method)
- Fetch the outputs from the buffer
- Check for status and error messages

After the Business Interlink object is instantiated, you can assign values from constants, PeopleSoft variables, or record fields to the inputs of that Business Interlink object.

When you execute the Business Interlink object, it loads the appropriate Business Interlink Plug-in and passes itself to that Business Interlink Plug-in. The Business Interlink Plug-in processes the input data, passing the input values of the Business Interlink object to the external system and then fills the output values of the Business Interlink object (if there are outputs).

Deciding Which Methods to Use

After you create your Business Interlink definition, you must use PeopleCode to instantiate an Interlink object and execute the Business Interlink plug-in. This PeopleCode can be long and complex. Rather than write it directly, you can drag and drop the Business Interlink definition from the Application Designer Project View into an open PeopleCode edit pane. Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify.

In this section, we discuss how to:

- Execute the Business Interlink object.
- Support batch input and output.
- Support rowsets.
- Use the flat table methods.
- Support dynamic output.
- Use hierarchical data (BIDocs).

See Also

See additional Business Interlink documentation available on PeopleSoft Customer Connection.

Executing the Business Interlink Object

In most cases, you must use the Execute method to execute the Business Interlink object. However, for bulk input, you can use the BulkExecute method instead.

The Execute and BulkExecute methods return a value you can use for status and error checking.

Supporting Batch Input and Output

The methods discussed in this section, except for BulkExecute, add input values to the Business Interlink object one set, or row, at a time. The call to the Business Interlink Plug-in occurs only once. All the input data is passed with the single Execute. All output is returned as batch as well. The methods then get the output values one set, or row, at a time.

If you're sending a large amount of data to the input buffers, instead of adding one input row at a time, you might write the data to a staging table, then use the BulkExecute method. This method automatically executes; that is, you don't have to use the Execute method. It also automatically fills the output record specified with the method with all the output values in every row in the output buffer if you've specified an output record.

Supporting Rowsets

If your data is mapped into rowsets, you may want to use the InputRowset method. This method takes a standard rowset object to populate the inputs for the Business Interlink object. You can use the FetchIntoRowset method to repopulate the rowset with new data.

Using the Flat Table Methods

If your data is in a flat table structure, you can use the flat table methods. AddInputRow adds rows of input to the Business Interlink object; FetchNextRow fetches rows of output from the Business Interlink object.

Supporting Dynamic Output

A Business Interlink can have dynamic output, meaning that the outputs for a Business Interlink object are changeable in data type or number of outputs.

Business Interlinks supplies a set of methods to support dynamic output. These methods enable you to interrogate the output buffer programmatically to determine the number of fields (columns), their types, and their values.

The following methods support dynamic output:

- GetFieldCount

- GetFieldType
- GetFieldValue
- MoveFirst
- MoveNext

Use the MoveFirst method to move to the first column, first row of the output buffer, and within a loop, use the MoveNext method to move to each row on the output buffer.

Within a MoveFirst (or MoveNext) loop, use the GetFieldCount method to get the number of columns in the output buffer, which is also the number of outputs for this Business Interlink object. Then you can extract the outputs from the buffer in a loop.

Within the GetFieldCount loop, use GetFieldName to get the name of the output, GetFieldValue to get the value of the output (which will be returned as a string), and GetFieldType to get the type of the output (if the output is not a string type, you will convert it to this type).

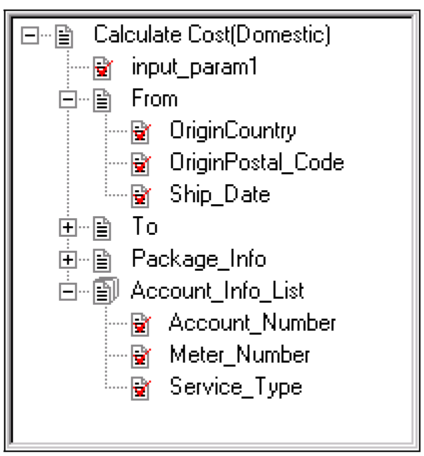
See Chapter 10, "Business Interlink Class," GetFieldCount, page 338; Chapter 10, "Business Interlink Class," GetFieldType, page 339; Chapter 10, "Business Interlink Class," GetFieldValue, page 340; Chapter 10, "Business Interlink Class," MoveFirst, page 352 and Chapter 10, "Business Interlink Class," MoveNext, page 353.

Using Hierarchical Data (BIDocs)

A Business Interlink can have hierarchical data. Think of the structure as a tree, with a root doc, node docs, and values. The hierarchical data methods and objects are also referred to as BIDocs.

Input Structures

The following shows an example of an input structure:



Example input structure

The *root doc* is Calculate Cost(Domestic). A root doc can contain both values and node docs.

The *node docs* are From, To, Package_Info and Account_Info_List. Each node can contain both values and child nodes. Node docs can be further described as either:

- *Simple node docs* have only one set of values for a single instance of a root doc. From, To, Package_Info are all simple node docs.
- *List node docs* can contain more than one set of values for a single instance of a root doc. Account_Info_List is a list node doc.

The values in the From node are OriginCountry, OriginPostal_Code, and Ship_Date. The value within the root node is input_param1. Notice that there are values both within the root doc and within node docs.

Business Interlinks support hierarchical input structures with the following methods:

- GetInputDocs
- AddDoc
- AddValue
- AddNextDoc

The GetInputDocs method returns a reference to the root doc of an input structure. From the previous example, it returns a reference to Calculate Cost(Domestic).

Use the AddDoc method to access the node docs. From the previous example, you would use AddDoc to access the From, To, Package_Info, and Account_Info_List node docs. If any of these nodes contained nodes, you could use AddDoc to access those as well.

Use the AddValue method to set values. From the previous example, you would use AddValue to set the value for input_param1, OriginCountry, OriginPosta_Code, and Ship_date. You must call AddDoc on a node before you can call AddValue for its values.

Use the AddNextDoc method to access the following:

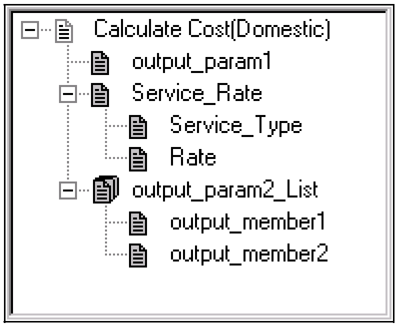
- If a node doc is a list, that is, it can contain more than one set of values, use AddNextDoc to reference the next set of values.
- To add another copy of the entire input structure, use AddNextDoc to return a reference to the next root doc.

The following code example sets values for the node doc From, which is a simple node doc. It also sets the values for Account_Info_List, which is a list node doc.

```
&Calc_Input = &QE_COST.GetInputDocs("");

&FromDoc = &Calc_Input.AddDoc("From");
&ret = &FromDoc.AddValue("OriginCountry", "United States");
&ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
&ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

&Account_Doc = &Calc_Input.AddDoc("Account_Info_List");
&ret = &Account_Doc.AddValue("Account_Number", "CT-8001");
&ret = &Account_Doc.AddValue("Meter_Number", &METER);
&ret = & Account_Doc.AddValue("Service_Type", &MODE);

/* add next set of values in list */

&ret = &Calc_Input.AddNextDoc();
&ret = &Account_Doc.AddValue("Account_Number", "CT-8002");
&ret = &Account_Doc.AddValue("Meter_Number", &METER);
&ret = &Account_Doc.AddValue("Service_Type", &MODE);
```


Output Structures

The following shows an example of an output structure:



Example output structure

The *root doc* is Calculate Cost(Domestic).

The *node docs* are Service_Rate and output_param2_List. Each node can contain both values and child nodes. Node docs can be further be described as either:

- *Simple node docs* have only one set of values for a single instance of a root doc. Service_Rate is a simple node doc.
- *List node docs* can contain more than one set of values for a single instance of a root doc. output_param2_List is a list node doc.

The values in the Service_Rate node are Service_Type, Rate, and in the output_param2_List node, output_member1 and output_member2, and within the root node, output_param1.

Business Interlinks support hierarchical output structures with the following methods:

- GetOutputDocs
- GetDoc
- GetNextDoc
- GetPreviousDoc
- GetStatus
- GetValue
- GetCount
- MoveToDoc

The GetOutputDocs method returns a reference to the root doc of an output structure. From the previous example, it returns a reference to Calculate Cost(Domestic).

Use the GetDoc method to access node docs. From the previous example, you would use GetDoc to access the Service_Rate or output_param2_List node docs. If any of these nodes contain nodes, you use GetDoc to access those as well.

Use the GetValue method to retrieve the values. From the previous example, you would use GetValue to retrieve the values for output_param1, and for the Service_Rate node, to get the values Service_Type, Rate, output_member1, and output_member2. You must call GetDoc on a node before you can call GetValue for its values.

Use the GetNextDoc (or GetPreviousDoc) method to access the following:

- A reference to the next (or previous, respectively) root doc.
- If a node doc is a list, that is, can contain more than one set of values, use GetNextDoc to reference the next node doc in the list (or GetPreviousDoc to access the previous node doc in the list.)

Use the GetCount method to return either the number of docs in a list node doc, or the number of root docs. In the example, you can count the number of Calculate Cost(Domestic) nodes or the number of output_param2_list nodes.

Use the MoveToDoc method to move to a particular doc at the root level or to a list node doc. In the example, you can move to a Calculate Cost(Domestic) node or to an output_param2_list_node.

The following code example gets values for the node docs Service Rate, which is a simple node doc. It also sets the values for output_param2_List, which is a list node doc.

```
&Calc_Input = &QE_COST.GetOutputDocs("");

&Service_Rate_Doc = &Calc_Input.GetDoc("Service_Rate");
&ret = &Service_Rate_Doc.GetValue("Service_Type", "Overnight");
&ret = &Service_Rate_Doc.GetValue("Rate", "50.00");

&Out_Param_Doc = &Calc_Input.GetDoc("output_param2_List");
&ret = &Out_Param_Doc.GetValue("output_member1", "value1");
&ret = &Out_Param_Doc.GetValue("output_member2", "value2");

/* get next set of values in list */

&Account_Doc = &Out_Param_Doc.AddNextDoc();
&ret = &Out_Param_Doc.GetValue("output_member1", "value3");
&ret = &Out_Param_Doc.GetValue("output_member2", "value4");
```

Using the Incoming Business Interlink Methods and Properties

The incoming Business Interlink methods enable you to parse an XML request and build an XML response.

The incoming Business Interlink uses a set of methods and functions.

The GetContentBody Request object method converts the request content into an XML string. You can then use this string with the GetBiDoc function to create a BiDocs structure from it that is the same shape and contains the same data as the XML document contained in the XML string.

After you have the BiDocs structure containing the XML request, you can:

- Use the GetNode method to get one of the XML elements.
- Use the NodeType method to get the type of the node (element, processing instruction, comment).
- Use the NodeName property to get the name of the element.

- Use the NodeValue property to get the value of the element.

You can also use the standard BiDocs methods (such as GetDoc, GetValue) to retrieve information from this BiDocs object.

When an XML element contains attributes, the AttributeCount property gets the number of attributes, the GetAttributeName method gets the name of an attribute, and the GetAttributeValue method gets the value of an attribute.

To build an XML response, you can use the GetBiDocs function to create a blank BiDocs structure. To create the XML structure within that BiDocs, use CreateElement to create an XML tag, AddComment to add an XML comment, AddAttribute to add an attribute to an XML tag, and AddProcessInstruction to add a processing instruction (the first tag of the XML response).

Use the GenXMLString method to create an XML string from the BiDocs structure. Then you can use the Write Response method to write the response to the HTTP response string.

See Also

See additional Business Interlink documentation available on PeopleSoft Customer Connection.

Understanding the State of an Interlink Object

PeopleSoft Business Interlink API that are run on the application server should be stateless, that is, if you want to save information from one call of the Interlink object to the next, you have to do it yourself by writing the relevant information to the database. If you use the Execute method more than once within a single PeopleCode event (that is, if you have the Execute method in some sort of loop) the state is preserved. After you leave the event, any state associated with the Interlink object is lost.

You should create only one Business Interlink object, that is, you should only use the GetInterlink function once. After that, you can load it with data, pass the data to the Interlink plug-in (using Execute) and fetch output data as many times as you need.

Using Business Interlink with Application Engine

In a regular PeopleCode program, you can only declare a Business Interlink object as local. However, in an Application Engine program, you can declare a Business Interlink object as global. Instantiating a Business Interlink object once as a global saves on the significant overhead of reinstantiating a local object for every iteration of PeopleCode called in a loop.

- Global Business Interlink objects can only be used in Application Engine PeopleCode programs because PeopleCode that runs on an application server must be stateless.
- When a restartable Application Engine program abends, global Business Interlink objects that were instantiated before the last checkpoint are automatically reinstantiated at restart. So the object will be available, even though no call has been made to GetInterlink in the restarted process. However, the associated Business Interlink data buffers are *not* recovered, so the Application Engine program must be written such that these buffers are empty whenever a checkpoint is taken.

- Business Interlink objects should *not* be declared as global unless they are used in several PeopleCode actions, or in a PeopleCode action that is called in a loop. Only in these instances is the overhead of doing a checkpoint worthwhile.

Using the Data Type of a Business Interlink Object

You should declare a Business Interlink object using the data type Interlink. For example,

```
Local Interlink &MYBI;
```

Declare hierarchical data as type BIDocs. For example:

```
Local BIDocs &OutlistDoc;
```

Note. BIDocs and Interlink objects used in PeopleCode programs run on the application server can be declared only as type Local. You can declare Interlinks as Global only in an Application Engine program.

Understanding the Scope of a Business Interlink Object

A Business Interlink object can be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

Business Interlink Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetBiDoc

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetInterlink

Business Interlink Class Methods

In this section, we discuss the Business Interlink class methods. The methods are discussed in alphabetical order.

AddDoc

Syntax

AddDoc*(docname)*

Description

The AddDoc method adds a document to an input structure. The added document is an input parameter for a Business Interlink object that is not of simple type (such as integer or string). You must add the document to the input structure before you can add values to its members with AddValue.

Parameters

Parameter	***Description***
docname	The name of the document that AddDoc adds to the structure.

Returns

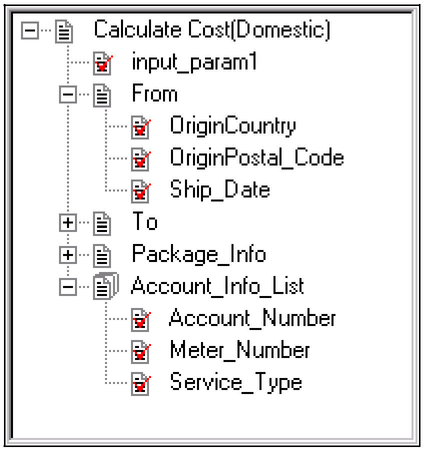
The document added to the BIDocs input document.

Example

In the following example, the input structure for Calculate Cost, or the root level document, is created with the GetInputDocs method. (To create, or add, more input structures, use AddNextDoc.) The Calculate Cost input parameter From is a document, so the AddDoc method adds it to the input document.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn;
Local BIDocs &FromDoc, &ToDoc, &PackageDoc, &AccountDoc;
Local number &ret, &retinput;


&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&CalcCostIn = &QE_COST.GetInputDocs("");

&ret = &CalcCostIn.AddValue("input_param1","value");
&FromDoc = &CalcCostIn.AddDoc("From");
&ret = &FromDoc.AddValue("OriginCountry", "United States");
&ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
&ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

/* Call AddDoc and AddValue for To, Package_Info, and Account_Info_List (code not⇒
 shown) */
```

The following shows the input structure for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List. This is a version of the Federal Express plug-in that was modified for this example (input_param1 was added, Account_Info was modified to be a list).



Example input structure

See Also

Chapter 10, "Business Interlink Class," GetInputDocs, page 341; Chapter 10, "Business Interlink Class," AddValue, page 317 and Chapter 10, "Business Interlink Class," AddNextDoc, page 314

AddInputRow

Syntax

`AddInputRow`*`(inputname, value)`*

where *inputname* and *value* are in matched pairs, in the form:

inputname1,*value1* [, *inputname2*, *value2*] . . .

Description

The AddInputRow method adds a row of input data *(value)* from PeopleCode variables or record fields for the Business Interlink object executing the method. These must be entered in matched pairs, that is, every input name must be followed by its matching value.

Note. The input *name*, not the input path, of the interface definition is used for this method.

There must be an input name for every input parameter defined in the interface definition used to instantiate the Business Interlink object.

If you specify a record field that is not part of the record the PeopleCode program is associated with, you must use *recname*..*fieldname* for that value.

You can specify default values for every input name in the interface definition (created in Application Designer.) These values are used if you create a PeopleCode "template" by dragging the interface definition from the Project window in Application Designer to an open PeopleCode editor window.

Parameters

Parameter	Description
inputname	Specify the input name. There must be one *inputname* for every input name defined in the interface definition used to instantiate the Business Interlink object.
value	Specify the value for the input name. This can be a constant, a variable, or a record field. Each *value* must be paired with an *inputname.*

Returns

A Boolean value: True if the input values were successfully added. Otherwise, it returns False.

Example

In the following example, the Business Interlink object name is SRA_ALL_1, and the input name, such as ship_site_name, are being bound to record fields, such as QE_RP_SITENAME or VENDOR_INFO.QE_RP_PROMISEDATE, to variables like &PARTNAME, or to literals, like 10 for quantity.

```
&SRA_ALL_1.AddInputRow("ship_site_name", QE_RP_SITENAME,
   "promise_date", VENDOR_INFO.QE_RP_PROMISEDATE,
   "request_date", QE_RP_ORDERREQDATE,
   "subline_site_name", QE_RP_SITENAME,
   "quantity", 10,
   "part_name", &PARTNAME,
   "site_name", INV_LOCATIONS.QE_RP_SITENAME);
```

See Also

Chapter 10, "Business Interlink Class," Execute, page 325

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetInterlink

AddNextDoc

Syntax

AddNextDoc()

Description

The AddNextDoc method adds a document to one of the following levels:

- The root level of the input structure for a Business Interlink object. You create a reference to this level with the GetInputDocs method.
- When a document within the input structure is a list, AddNextDoc adds another document to the list. If you use AddNextDoc on a document that is not a list, AddNextDoc fails and returns an error.

Parameters

None.

Returns

This method returns an integer. The values are:

Value	*Description*
0	eNoError. The method succeeded.
1	eNO_DOCUMENT. The document referenced by this method does not exist.
2	eDOCLIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED. Internal error.
4	eNOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	eNOT_LISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE. You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA. You tried to retrieve data from a document that contained no data.

Example

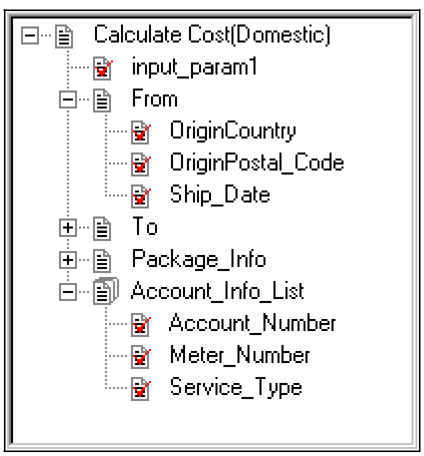
In this example, the input structure for Calculate Cost, or the root level document, is accessed with the GetInputDocs method. The AddNextDoc method adds another BIDocs input document. The Calculate Cost input parameter Account_Info_List is a document, so the AddDoc method adds it to the BIDocs input document. Account_Info_List is also a document list, so AddNextDoc method adds another Account_Info_List document.


```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn;
Local BIDocs &FromDoc, &ToDoc, &PackageDoc, &AccountDoc;
Local number &ret, &retinput;


&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&CalcCostIn = &QE_COST.GetInputDocs("");

For &n = 1 to &number_of_input_sets
/* Get values for inputs, such as &ORIGIN (code not shown) */

   &ret = &CalcCostIn.AddValue("input_param1","value");
   &FromDoc = &CalcCostIn.AddDoc("From");
   &ret = &FromDoc.AddValue("OriginCountry", "United States");
   &ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
   &ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

   /* Call AddDoc and AddValue for To and Package_Info
      (code not shown) */

   /* Call AddDoc and AddNextDoc for the AccountDoc document list */
   &AccountDoc = &CalcCostIn.AddDoc("Account_Info_List");
   For &m = 1 to &number_of_AccountDocs
      &ret = &AccountDoc.AddValue("Account_Number", &ACCOUNT);
      &ret = &AccountDoc.AddValue("Meter_Number", &METER);
      &ret = &AccountDoc.AddValue("Service_Type", &SVCTYPE);
      &retinput = &AccountDoc.AddNextDoc();
   End-For;

   &retinput = &CalcCostIn.AddNextDoc();

End-For;
```

The following shows the input structure for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List.

From, To, and Package_Info are not lists, so if you try to use AddNextDoc with these parameters, such as the following line of code, AddNextDoc will fail and return an error message.

```
&retinput = &To.AddNextDoc();
```



Example input structure

See Also

Chapter 10, "Business Interlink Class," GetInputDocs, page 341; Chapter 10, "Business Interlink Class," AddDoc, page 311 and Chapter 10, "Business Interlink Class," AddValue, page 317

AddValue

Syntax

`AddValue``(`*`paramname, value`*`)`

Description

The AddValue method adds a value to a member of a document within the input structure for a Business Interlink object.

Parameters

Parameter	***Description***
paramname	The name of the member of the document that is having a value added to it.
value	The value that is added. This can be a constant, a variable, or a record field. The data type can be string, integer, double, float, time, date, or datetime.

Returns

This method returns an integer. The values are:

Value	***Description***
0	eNoError. The method succeeded.
1	eNO_DOCUMENT. The document referenced by this method does not exist.
2	eDOCLIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED. Internal error.
4	eNOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.

Value	Description
5	eNOT_LISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE. You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA. You tried to retrieve data from a document that contained no data.

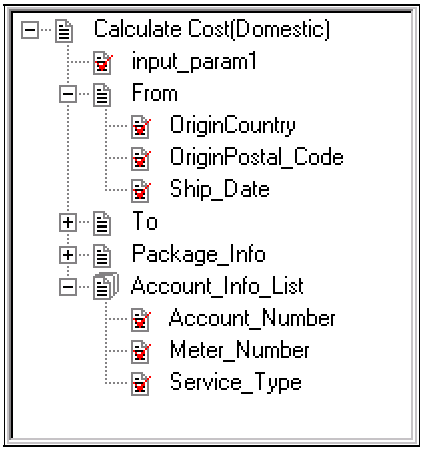
Example

In the following example, the Business Interlink object name is QE_COST.

In the following example, the input structure for Calculate Cost, or the root level document, is accessed with the GetInputDocs method. (To create, or add, more input structures, use AddNextDoc.) The Calculate Cost input parameter From is a document, so the AddDoc method adds it to the input structure. Then the AddValue method adds values to each of the From document members.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn;
Local BIDocs &FromDoc, &ToDoc, &PackageDoc, &AccountDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

/* Get some values for input, such as &ORIGIN (code not shown) */

&CalcCostIn = &QE_COST.GetInputDocs("");
&ret = &CalcCostIn.AddValue("input_param1","value");

&FromDoc = &CalcCostIn.AddDoc("From");
&ret = &FromDoc.AddValue("OriginCountry", "United States");
&ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
&ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

/* Call AddDoc, AddValue for Package, Account_Info_List, and also call AddNextDoc⇒
 for Account_Info_List (code not shown) */
```

The following shows the input structure for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List.



Example input structure

See Also

Chapter 10, "Business Interlink Class," GetInputDocs, page 341; Chapter 10, "Business Interlink Class," AddDoc, page 311 and Chapter 10, "Business Interlink Class," AddNextDoc, page 314

BulkExecute

Syntax

BulkExecute(RECORD.*inputrecname* [, **RECORD.***outputrecname*] [, {*user_process_inst* | *user_operid*}])

Description

The BulkExecute method uses the data in the specified record to populate the input buffer, copying like-named fields. Then the method executes, and, optionally, fills the record specified by outputrecname with data from the Business Interlink output buffer. BulkExecute results in significantly faster performance for transactions that process large amounts of data. Instead of adding one input row at a time, then fetching the values one at a time, you might write the data to a staging table, use the BulkExecute method and then read the data from the output table. This would be especially effective in Application Engine programs that process sets of data rather than individual rows.

This method assumes that the names of the fields in the record match the names of the inputs (or outputs) defined in the Business Interlink Definition. If there is no field in the *inputrecname* for a Business Interlink input parameter, the parameter's default value is used. If no default is specified, an empty string is passed. You must ensure that this default value is legitimate.

Fields in the output buffer are matched against the fields specified in the output record. You do not have to specify an output record. If you don't specify an output record, the output buffers are not be populated.

If you specify an output record, and if you have fields in the output record that are not specified as output parameters, they aren't populated with the BulkExecute method. In addition, they shouldn't be set as NOT NULL, otherwise inserts fail.

Note. Before you use this method, you should flush the record used for output and remove any residual data that might exist in it.

If you specify an output record, and you want to use the output record for error checking, you must add the following fields to your output record:

- RETURN_STATUS
- RETURN_STATUS_MSG

Note. The field names in the output record must match these names exactly.

Then you must mark one or more input parameters as "key fields" in the Business Interlink definition (using the BulkExecute ID check box.) The value of these key fields are copied to the output record, and you can use these fields to match error messages with input (or output) rows.

To order your input (and output) rows, you must add the following column to both the input and output table:

- BI_SEQ_NUM
-

Note. The field name in your input and output records must match this name exactly.

The input rows are read in the order of numbers in BI_SEQ_NUM, and the output rows are generated using the same order number.

This method automatically executes, so you don't need to use the Execute method with this method.

Whether this method halts on execution depends on the setting of the StopAtError configuration parameter. The default value is True, that is, stop if the error number returned is something other than a 1 or 2. This configuration parameter must be set before using the BulkExecute method.

Parameters

Parameter	***Description***
RECORD.*inputrecname*	Specify a record (SQL table) that contains the data to populate the input buffer of the Business Interlink.
RECORD.*outputrecname*	Specify a record (SQL table) that will hold the data that populates the output buffer of the Business Interlink. This is optional; it is often used to error check the input.
user_process_inst | *user_operid*	This is an optional parameter that allows either different Application Engine programs or different clients to populate the same **RECORD.***outputrecname* at the same time.
	For *user_process_inst*, the parameter takes an integer. **RECORD.***outputrecname* must have a PROCESS_INSTANCE field. The PROCESS_INSTANCE field is used to identify the Application Engine program that is using this Business Interlink. You can use the %PROCESS_INSTANCE variable to populate *user_process_inst.*

Parameter	**Description**
	For *user_operid,* the parameter takes a string. **RECORD.***outputrecname* must have an OPERID field. The OPERID field is used to identify the client who is using this Business Interlink. You can use the %OPERID variable to populate *user_operid.*

Returns

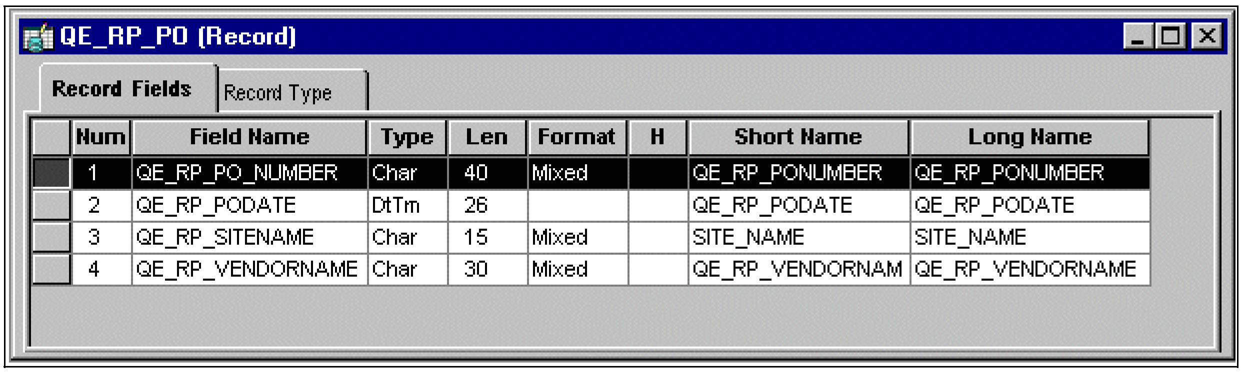
The following are the valid returns:

Value	**Description**
1	The Business Interlink object executes successfully
2	The Business Interlink object failed to execute
3	Transaction failed
4	Query failed
5	Missing criteria
6	Input mismatch
7	Output mismatch
8	No response from server
9	Missing parameter
10	Invalid username
11	Invalid password
12	Invalid server name
13	Connection error
14	Connection refused
15	Timeout reached
16	Unequal lists
17	No data for output
18	Output parameters empty
19	Driver not found

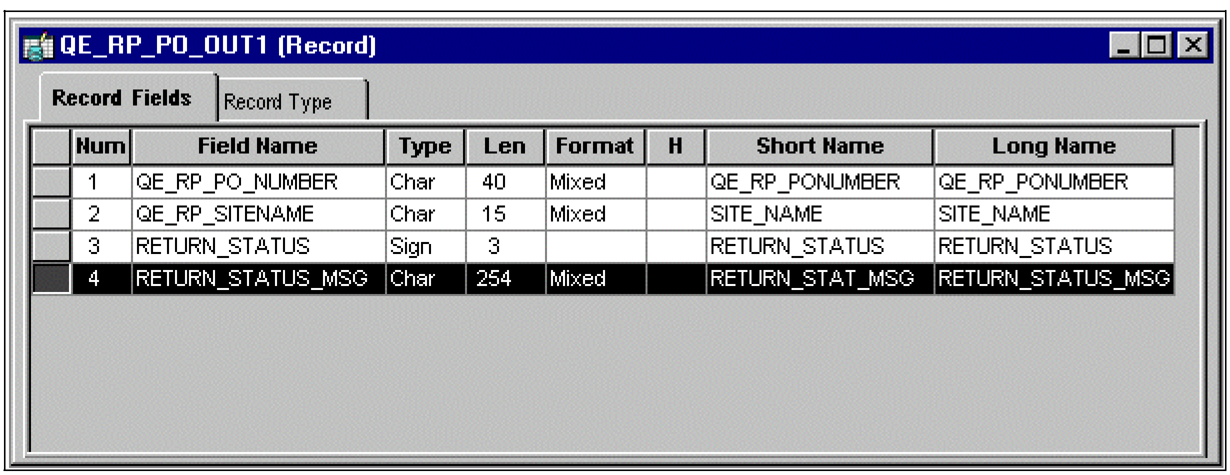
Value	***Description***
20	Internet connect error
21	XML parser error
22	XML deserialize

Example

Here are two PeopleSoft records that could be used as input and output records for BulkExecute. The key fields in the input record, which need to have the BulkExecute ID check box checked in Application Designer, are QE_RP_PO_NUMBER and QE_RP_SITENAME. These key fields are used both for input and output.

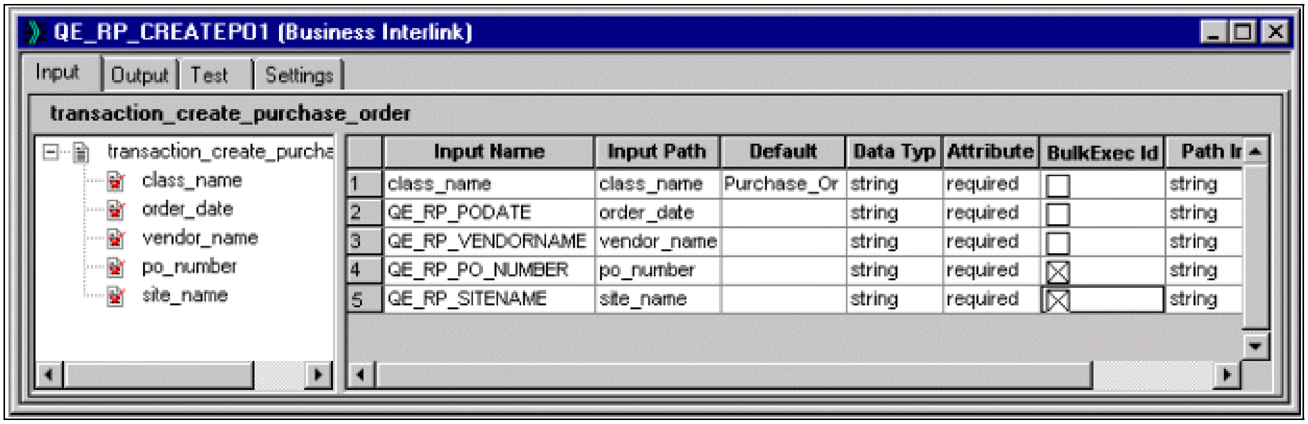


Example input record for BulkExecute

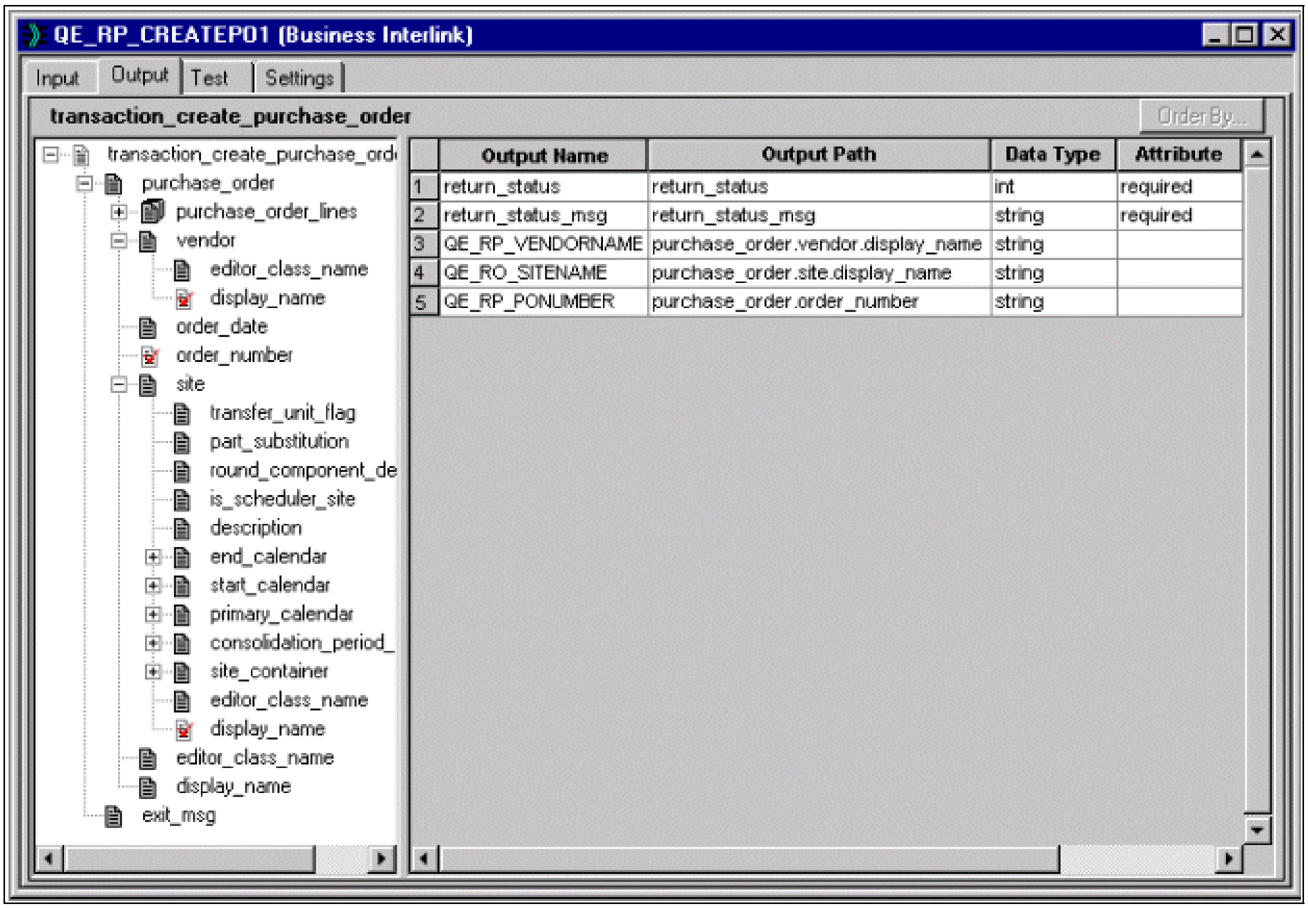


Example output record for BulkExecute

Here are the corresponding inputs and outputs for a Business Interlink that has had its inputs and outputs renamed to match the field names in the records. The inputs and outputs have been renamed where necessary to match the record field names.



Example Business Interlink inputs for BulkExecute



Example Business Interlink inputs for BulkExecute

The PeopleCode program using these records could contain the following code to run BulkExecute.


```
Local Interlink &CREATE_PO;
Local number &EXECRSLT;

&CREATE_PO = GetInterlink(INTERLINK.QE_RP_CREATEPO1);

/* The next three lines set configuration parameters, which are not shown in the⇒
 previous examples. */
&CREATE_PO.SERVER_NAME = "bc1";
&CREATE_PO.RSERVER_HOST = "4.3.4.3";
&CREATE_PO.RSERVER_PORT = "2200";

&EXECRSLT = &CREATE_PO.BulkExecute(RECORD.QE_RP_PO, RECORD.QE_RP_PO_OUT1);
```

See Also

Chapter 10, "Business Interlink Class," StopAtError, page 372

Clear

Syntax

Clear()

Description

The Clear method clears the input and output buffers. If you're using Business Interlinks in a batch program, every time after you use the Execute method, use the Clear method to flush out the input and output buffers.

Parameters

None.

Returns

None.

Example

```
For &n = 1 to x
   &myinterlink.AddInputRow("input1", value1, "input2", value2);
   &myinterlink.Execute();
   &myinterlink.FetchNextRow("output1", value1, "output2", value2);
      /* do processing on data */
   &myinterlink.Clear();
   &n = &n + 1;
End-for;
```


See Also

Chapter 10, "Business Interlink Class," Execute, page 325 and Chapter 10, "Business Interlink Class," BulkExecute, page 319

Execute

Syntax

`Execute``()`

Description

When an Interlink object executes the Execute method, the transaction associated with the Interlink object is executed.

If there is only one row, after appropriate substitution is made, the transaction is executed only once. Otherwise, the data is "batched up" and sent once. You have to call Execute only once to execute all the rows of the input buffer.

Generally, you would only use the Execute method after using the AddInputRow method. If you create a PeopleCode "template" by dragging the Business Interlink definition from the Project window in Application Designer to an open PeopleCode editor window, the usual order of the execution of methods is shown in the template.

If you're using Business Interlinks in a batch program, every time after you use the Execute method, use the Clear method to flush out the input and output buffers.

Whether this method halts on execution depends on the setting of the StopAtError configuration parameter. The default value is True, that is, stop if the error number returned is something other than a 1 or 2. This configuration parameter must be set *before* using the Execute method.

Parameters

None.

Returns

The following are the valid returns:

Value	*Description*
1	The Business Interlink object executes successfully
2	The Business Interlink object failed to execute
3	Transaction failed

Value	Description
4	Query failed
5	Missing criteria
6	Input mismatch
7	Output mismatch
8	No response from server
9	Missing parameter
10	Invalid username
11	Invalid password
12	Invalid server name
13	Connection error
14	Connection refused
15	Timeout reached
16	Unequal lists
17	No data for output
18	Output parameters empty
19	Driver not found
20	Internet connect error
21	XML parser error
22	XML deserialize

Example

```
Local number &EXECRSLT;
.  .  .
&EXECRSLT = &SRA_ALL_1.Execute();
If (&EXECRSLT <> 1) Then
/* The instance failed to execute */
/* Do Error Processing */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," BulkExecute, page 319 and Chapter 10, "Business Interlink Class," Clear, page 324

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetInterlink

FetchIntoRecord

Syntax

```
FetchIntoRecord(RECORD.recname, [, {user_process_inst | user_operid}])
```

Description

The FetchIntoRecord method copies the data from the output buffer into the specified record (SQL table) copying *like-named* fields. This method assumes that the names of the fields in the record match the names of the outputs defined in the Business Interlink definition.

You can use the FetchIntoRecord method to perform a transaction on a large amount of data that you want to receive from your system. Instead of executing your Business Interlink and then fetching one output row at a time, you might execute your Business Interlink, then use the FetchIntoRecord method to write the data returned from the Business Interlink to a staging table.

Note. Before you use this method, you should flush the record used for output and remove any residual data that might exist in it.

If your data is hierarchical, that is, in a rowset, and you want to preserve the hierarchy, use FetchIntoRowset instead of this method.

To order your output rows, you must add the BI_SEQ_NUM column to the record.

This column is populated with a sequence number that corresponds to the order in which each row of the record was written to by the method.

Parameters

Parameter	*Description*
RECORD.*recname*	Specify a record (SQL table) that you want to populate with data from the output buffers.

Parameter	***Description***
user_process_inst | *user_operid*	This is an optional parameter that enables either different Application Engine programs or different clients to populate the same **RECORD.***outputrecname* at the same time. For *user_process_inst*, the parameter takes an integer. **RECORD.***outputrecname* must have a PROCESS_INSTANCE field. The PROCESS_INSTANCE field is used to identify the Application Engine program that is using this Business Interlink. You can use the %PROCESS_INSTANCE variable to populate *user_process_inst.* For *user_operid,* the parameter takes a string. **RECORD.***outputrecname* must have an OPERID field. The OPERID field is used to identify the client who is using this Business Interlink. You can use the %OPERID variable to populate *user_operid.*

Returns

Number of rows fetched if one or more non-empty rows are returned.

If an empty row is returned, that is, every if field is empty except the return_status and return_status_msg fields, this method returns one of the following error messages:

Number	***Meaning***
0	No error
1	NoInterfaceObject
2	ParamCountError
3	IncorrectParameterType
4	NoColumnForOutputParm
5	NoColumnForInputParm
6	BulkInsertStartFailed
7	BulkInsertStopFailed
8	CouldNotCreateSelectCursor
9	CouldNotCreateInsertCursor

Number	*Meaning*
10	CouldNotDestroySelectCursor
11	CouldNotDestroyInsertCursor
12	InputRecordDoesNotExist
13	OutputRecordDoesNotExist
14	ContainEmptyRow
15	SQLExecError
201	FieldTooLarge
202	IgnoreSignNumber
203	ConvertFloatToInt

Example

```
&RSLT = &QE_SM_CONCAT.FetchIntoRecord(RECORD.PERSONAL__VENDR_DATA);

If &RSLT = 0 Then
/* no rows fetched */
Else
   /* do processing */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," BulkExecute, page 319

FetchIntoRowset

Syntax

FetchIntoRowset*(&Rowset)*

Description

The FetchIntoRowset method copies the data from the output buffer into the specified rowset, copying *like-named* fields. This method assumes that the names of the fields in the rowset match the names of the outputs defined in the Business Interlink definition, and that the structure is the same.

Note. Before you use this method, you should flush the rowset used for output and remove any residual data that might exist in it.

Use this method only if you have a hierarchical data structure and you want to preserve the hierarchy. Otherwise, use BulkExecute or FetchIntoRecord.

Parameters

Parameter	*Description*
&Rowset	Specify rowset object that you want to populate with data from the output buffers. This must be an existing, instantiated rowset.

Returns

Number of rows fetched if one or more non-empty rows are returned.

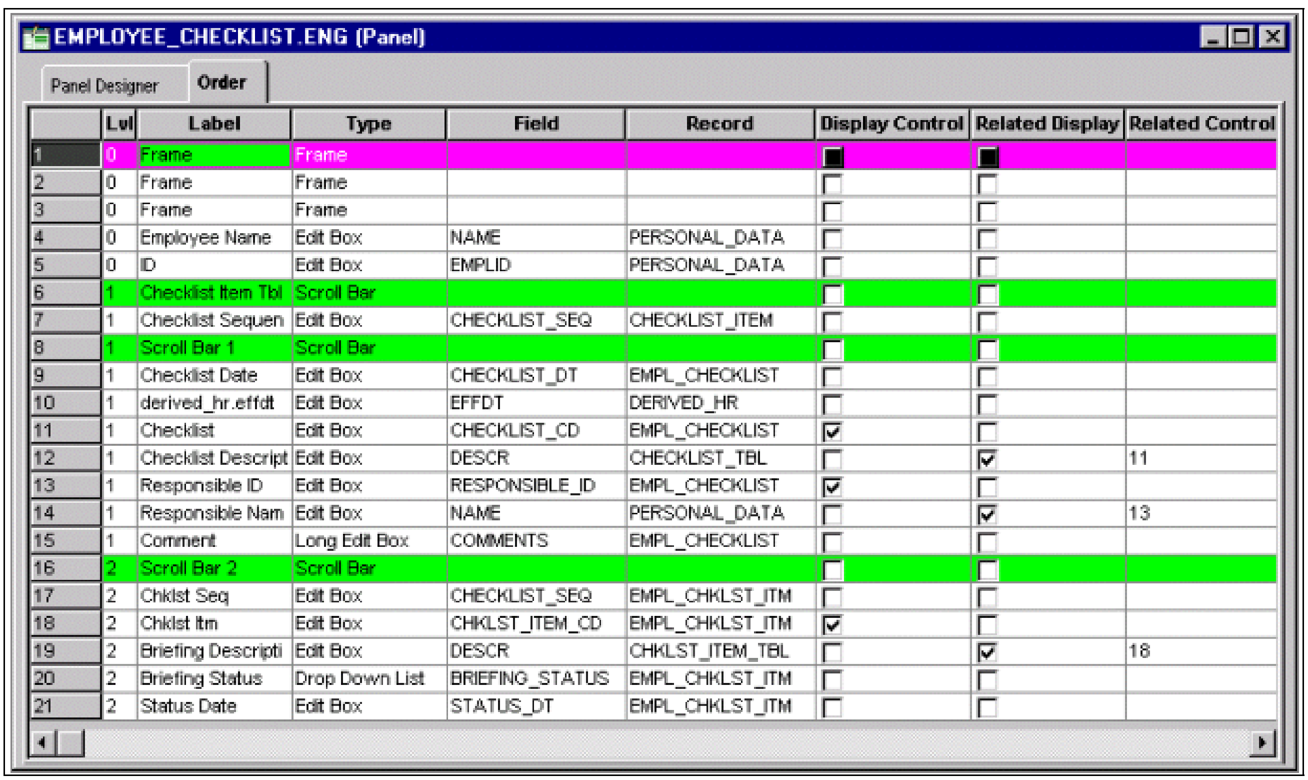
If an empty row is returned, that is, every if field is empty except the return_status and return_status_msg fields, this method returns one of the following error messages:

Value	*Description*
0	No error
1	NoInterfaceObject
2	ParamCountError
3	IncorrectParameterType
4	NoColumnForOutputParm
5	NoColumnForInputParm
6	BulkInsertStartFailed
7	BulkInsertStopFailed
8	CouldNotCreateSelectCursor
9	CouldNotCreateInsertCursor

Value	*Description*
10	CouldNotDestroySelectCursor
11	CouldNotDestroyInsertCursor
12	InputRecordDoesNotExist
13	OutputRecordDoesNotExist
14	ContainEmptyRow
15	SQLExecError
201	FieldTooLarge
202	IgnoreSignNumber
203	ConvertFloatToInt

Example

The example uses the rowset on level one from the EMPLOYEE_CHECKLIST page. A PeopleCode program running in a field on level zero in that page can access the level one (child rowset).



	Lvl	Label	Type	Field	Record	Display Control	Related Display	Related Control
1	0	Frame	Frame			■	■	
2	0	Frame	Frame			☐	☐	
3	0	Frame	Frame			☐	☐	
4	0	Employee Name	Edit Box	NAME	PERSONAL_DATA	☐	☐	
5	0	ID	Edit Box	EMPLID	PERSONAL_DATA	☐	☐	
6	1	Checklist Item Tbl	Scroll Bar			☐	☐	
7	1	Checklist Sequen	Edit Box	CHECKLIST_SEQ	CHECKLIST_ITEM	☐	☐	
8	1	Scroll Bar 1	Scroll Bar			☐	☐	
9	1	Checklist Date	Edit Box	CHECKLIST_DT	EMPL_CHECKLIST	☐	☐	
10	1	derived_hr.effdt	Edit Box	EFFDT	DERIVED_HR	☐	☐	
11	1	Checklist	Edit Box	CHECKLIST_CD	EMPL_CHECKLIST	☑	☐	
12	1	Checklist Descript	Edit Box	DESCR	CHECKLIST_TBL	☐	☑	11
13	1	Responsible ID	Edit Box	RESPONSIBLE_ID	EMPL_CHECKLIST	☑	☐	
14	1	Responsible Nam	Edit Box	NAME	PERSONAL_DATA	☐	☑	13
15	1	Comment	Long Edit Box	COMMENTS	EMPL_CHECKLIST	☐	☐	
16	2	Scroll Bar 2	Scroll Bar			☐	☐	
17	2	Chklst Seq	Edit Box	CHECKLIST_SEQ	EMPL_CHKLST_ITM	☐	☐	
18	2	Chklst Itm	Edit Box	CHKLST_ITEM_CD	EMPL_CHKLST_ITM	☑	☐	
19	2	Briefing Descripti	Edit Box	DESCR	CHKLST_ITEM_TBL	☐	☑	18
20	2	Briefing Status	Drop Down List	BRIEFING_STATUS	EMPL_CHKLST_ITM	☐	☐	
21	2	Status Date	Edit Box	STATUS_DT	EMPL_CHKLST_ITM	☐	☐	

EMPLOYEE_CHECKLIST Page order

EMPL_CHECKLIST is the primary database record for the level one scrollbar on the EMPLOYEE_CHECKLIST page. The following PeopleCode accesses the level one rowset using EMPL_CHECKLIST. The Business Interlink object name is QE_BI_EMPL_CHECKLIST. This Business Interlink object uses the level one rowset as its input and its output.

The InputRowset method uses this rowset as input for QE_BI_EMPL_CHECKLIST. Then a blank duplicate of the rowset is created with CreateRowset, and then the output of QE_BI_EMPL_CHECKLIST is fetched into the blank rowset with FetchIntoRowset.

```
&MYROWSET = GetRowset(SCROLL.EMPL_CHECKLIST);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.InputRowset(&MYROWSET);
&RSLT = &QE_BI_EMPL_CHECKLIST.Execute();
/* do some error processing */
&WorkRowset = CreateRowset(&MYROWSET);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.FetchIntoRowset(&WorkRowset);

If &ROWCOUNT = 0 Then
/* do some error processing */
Else
   /* Process the rowset from QE_BI_EMPL_CHECKLIST. Check it for errors. */
   For &I = 1 to &WorkRowset.RowCount
      For &K = 1 to &WorkRowset(&I).RecordCount
         &REC = &WorkRowset(&I).GetRecord(&K);
         &REC.ExecuteEdits();
         For &M = 1 to &REC.FieldCount
            If &REC.GetField(&M).EditError Then
            /* there are errors */
            /* do other processing */
            End-If;
         End-For;
      End-For;
   End-for;
End-if;
```

See Also

Chapter 10, "Business Interlink Class," InputRowset, page 350

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

FetchNextRow

Syntax

FetchNextRow*(outputname, value)*

where *outputname* and *value* are in matched pairs, in the form:

outputname1, value1 [*, outputname2, value2*] . . .

Description

After the Business Interlink object executes the method Execute, the FetchNextRow method can be used to retrieve a row of output and store the values of the output name *(outputname)to* PeopleCode variables or record fields *(value).* These must be entered in matched pairs, that is, every output name must be followed by its matching value.

Note. The output *name,* not the output path of the interface definition is used for this method.

There must be an *outputname* for every output name defined in the interface definition used to instantiate the Business Interlink object.

If you specify a record field that is not part of the record the PeopleCode program is associated with, you must use *recname.fieldname* for that *value.*

You can specify default values for every output name in the interface definition (created in Application Designer.) These values are used if you create a PeopleCode "template" by dragging the interface definition from the Project window in Application Designer to an open PeopleCode Editor window.

Parameters

Parameter	***Description***
outputname	Specify the output name. There must be one *outputname* for every output name defined in the interface definition used to instantiate the Business Interlink object.
value	Specify the value for the output name. This can be a constant, a variable, or a record field. Each *value* must be paired with an *outputname.*

Returns

A Boolean value: True if the row of output parameters was fetched. Otherwise, it returns False.

Example

In the following example, the Business Interlink object name is SRA_ALL_1, and the output names, such as costs, are being bound to record fields, such as STR_COST.

```
&RSLT = &SRA_ALL_1.FetchNextRow("costs", &STR_COST,
          "unit_costs", &STR_UNIT_COST,
          "customer_ship_dates", &STR_SHIP_DATE,
          "quantities", &STR_QUANTITY,
          "so_numbers", &STR_SO_NUM,
          "so_names", &STR_SO_NAME,
          "line_numbers", &STR_LINE_NUM,
          "ship_sets", &STR_SHIP_SET,
          "customer_receipt_dates", &STR_RECPT_DATE);
```


See Also

Chapter 10, "Business Interlink Class," Execute, page 325

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetInterlink

GetCount

Syntax

`GetCount``(`*`docname`*`)`

Description

The GetCount method returns the number of documents within a document list contained within an output structure for a Business Interlink object.

Parameters

Parameter	*Description*
docname	The name of the document list.

Returns

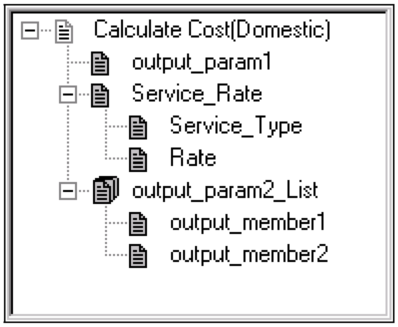
The number of documents in the list.

Example

In the following example, the output structure for Calculate Cost, or the root level document, is accessed with the GetOutputDocs method. The Calculate Cost output parameter output_param2_List is a document, so the GetDoc method gets it from the output structure. Because output_param2_List is a document list, GetCount gets the number of documents in the list.


```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
/* Get inputs, execute. (code not shown) */

&CalcCostOut = &QE_COST.GetOutputDocs("");

/* Call GetValue for output_param1, call GetDoc, GetValue for
Service_Rate (code not shown) */

&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
&count = &CalcCostOut.GetCount("output_param2_List");

&I = 0;
While (&I < &count)
   &ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
   &ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
   If &ret = 0 Then
      /* Process output values */
      &I = &I + 1;
      &retoutput = &OutlistDoc.GetNextDoc();
   End-If;
End-While;
```

The following shows the output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example output structure

GetDoc

Syntax

GetDoc*(docname)*

Description

The GetDoc method gets a document from an output structure. The document is an output parameter for a Business Interlink object that is not of simple type (such as integer or string). You must get the document from the output structure before you can get values from its members with GetValue.

You can call GetDoc using dot notation, to access documents that are "nested" within other documents. For example, the following accesses a document nested three levels deep:

```
&Docs3 = &CalcCostOut.GetDoc("Doc1.Doc2.Doc3");
```

Parameters

Parameter	***Description***
docname	The name of the document that GetDoc should get from the output structure.

Returns

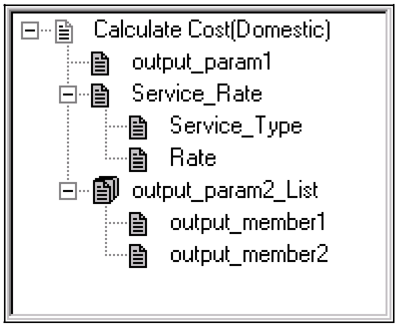
A reference to the document received from the output structure.

Example

In the following example, the Output structure for Calculate Cost(Domestic), or the root level document, is created with the GetOutputDocs method. (To create, or get, more Output structures, call GetNextDoc.) The Calculate Cost output parameter Service_Rate is a document, so the GetDoc method gets it from the Output structure.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &ServiceRateDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&ret = &CalcCostOut.GetValue("output_param1",&VALUE);
&ServiceRateDoc = &CalcCostOut.GetDoc("Service_Rate");
&ret = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
&ret = &ServiceRateDoc.GetValue("Rate", &RATE);
/* Call GetDoc, GetValue, GetNextDoc for output_param2_List (code not shown) */
If &ret = 0 Then
   /* Process output values */
End-If;
```

The following illustration shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Federal Express plug-in that was modified for this example (output_param1 and output_param2_List were added).



Example output structure

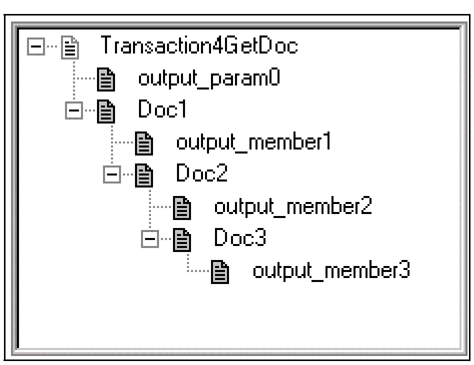
In the following example, GetDoc is used to access a document that is nested more deeply. To access a document that is more deeply nested, you can either call GetDoc for each nesting, or you can call GetDoc once using the nesting feature.

Calling GetDoc with the nesting feature:

```
Local Interlink &QE_4GETDOC;
Local BIDocs &CalcCostOut, &Docs3;
Local number &ret;

&QE_4GETDOC = GetInterlink(Interlink.QE_4GETDOC_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_4GETDOC.GetOutputDocs("");
&Docs3 = &CalcCostOut.GetDoc("Doc1.Doc2.Doc3");
&ret = &Docs3.GetValue("output_member3", &VALUE);
```

Calling GetDoc without the nesting feature:

```
Local Interlink &QE_4GETDOC;
Local BIDocs &CalcCostOut, &Docs1, &Docs2, &Docs3;
Local number &ret;

&QE_4GETDOC = GetInterlink(Interlink.QE_4GETDOC_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_4GETDOC.GetOutputDocs("");
&Docs1 = &CalcCostOut.GetDoc("Doc1");
&Docs2 = &Docs1.GetDoc("Doc2");
&Docs3 = &Docs2.GetDoc("Doc3");
&ret = &Docs3.GetValue("output_member3", &VALUE);
```



Example output structure with nested documents

GetFieldCount

Syntax

GetFieldCount()

Description

Use the GetFieldCount method to support dynamic output. It returns the number of columns in the output buffer, which is the same as the number of outputs in each row of the output buffer.

Note. This method can also be used with hierarchical doc data.

Parameters

None.

Returns

The number of columns in the output buffer, which is the same as the number of outputs in each row of the output buffer.

Example

The following example moves to the first column, first field of the output buffer. The Repeat loop goes through every row in the output buffer. The For loop processes every field in every row.

```
If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            /* do processing */
         .  .  .
         End-Evaluate;
      End-For;
   Until Not (&MYBI.MoveNext());
Else
   /* do error processing - output buffer not returned */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," GetFieldType, page 339; Chapter 10, "Business Interlink Class," GetFieldValue, page 340; Chapter 10, "Business Interlink Class," MoveFirst, page 352; Chapter 10, "Business Interlink Class," MoveNext, page 353 and Chapter 10, "Business Interlink Class," Using Hierarchical Data (BIDocs), page 306

GetFieldType

Syntax

`GetFieldType`*`(index)`*

Description

Use the GetFieldType method to support dynamic output. It returns the type of field specified by *index.* You can use this method only after you have used the MoveFirst method: otherwise the system doesn't know where to start.

Note. This method can also be used with hierarchical doc data.

Parameters

Parameter	*Description*
index	Specify the number of the field you want to find the type of.

Returns

A number indicating the type of the field. The values are:

Value	*Description*
1	String
2	Integer
3	Float
4	Boolean
5	Date
6	Time
7	DateTime
8	Binary
9	Object

Example

In the following example, the Business Interlink object name is &MYBI. The example uses MoveFirst to move to the first row of the output buffer, and to the first column, or first field, in that row. The Repeat loop uses MoveNext to go through every row in the output buffer. The For loop processes every field in every row, using the GetFieldCount method to get the number of fields, or outputs, in the row. Within the For loop, GetFieldType gets the type of the field data (string, integer, and so on.)

```
/* Add inputs to the Business Interlink object, then call Execute
to execute the Business Interlink object.
You are then ready to get the outputs using the following code. */

If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            &STRING_VARIABLE = &MYBI.GetFieldValue(&I);
         /* test for and process other field types */
         End-Evaluate;
      End-For;
   Until Not(&MYBI.MoveNext());
Else
   /* Process error - no output buffer */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," GetFieldCount, page 338; Chapter 10, "Business Interlink Class," GetFieldValue, page 340; Chapter 10, "Business Interlink Class," MoveFirst, page 352 and Chapter 10, "Business Interlink Class," MoveNext, page 353

GetFieldValue

Syntax

GetFieldValue*(index)*

Description

Use the GetFieldValue method to support dynamic output. It returns the value of the field specified by index. You can use this method only after you have used the MoveFirst method: otherwise the system doesn't know where to start.

Note. This method can also be used with hierarchical doc data.

Parameters

Parameter	*Description*
index	Specify the number of the field you want to find the value of.

Returns

A string containing the value of the field.

Example

In the following example, the Business Interlink object name is &MYBI. The example uses MoveFirst to move to the first row of the output buffer, and to the first column, or first field, in that row. The Repeat loop uses MoveNext to go through every row in the output buffer. The For loop processes every field in every row, using the GetFieldCount method to get the number of fields, or outputs, in the row. Within the For loop, GetFieldValue gets the value of the field data.

```
/* Add inputs to the Business Interlink object, then call Execute to execute
the Business Interlink object. You are then ready to get the
outputs using the following code. */

If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            &STRING_VARIABLE = &MYBI.GetFieldValue(&I);
         /* test for and process other field types */
         End-Evaluate;
      End-For;
   Until Not(&MYBI.MoveNext());
Else
   /* Process error - no output buffer */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," GetFieldCount, page 338; Chapter 10, "Business Interlink Class," GetFieldType, page 339; Chapter 10, "Business Interlink Class," MoveFirst, page 352 and Chapter 10, "Business Interlink Class," MoveNext, page 353

GetInputDocs

Syntax

GetInputDocs("")

Description

The GetInputDocs method gets the top input document at the root level for a Business Interlink object. This is a hierarchical structure that contains the values for the inputs for this Business Interlink object. The methods that you use to put the input values into this document are the hierarchical data methods.

Parameters

A null string.

Returns

An input document. This is the document at the top of the root level of the input document for a Business Interlink object.

Example

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostOut, &CalcCostIn;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
&CalcCostIn = &QE_COST.GetInputDocs("");

/* You can now insert the input values and execute the BI object */
```

See Also

Chapter 10, "Business Interlink Class," Using Hierarchical Data (BIDocs), page 306

GetNextDoc

Syntax

GetNextDoc()

Description

The GetNextDoc method gets a document from one of the following levels:

- The root level of the Output structure for a Business Interlink object. This level was accessed with the GetOutputDocs method.
- When a document within the Output structure is a list, GetNextDoc gets another document from the list. If you use GetNextDoc on a document that is not a list, GetNextDoc fails and returns an error.

Parameters

None.

Returns

The following are the valid returns:

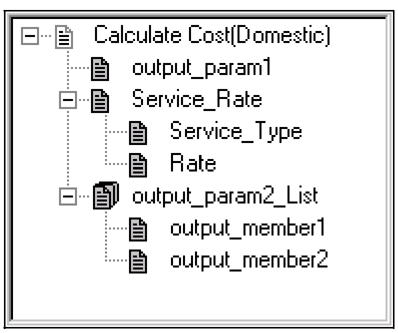
Number	***Enum Value***	***Description***
0	eNoError	The method succeeded.
1	eNO_DOCUMENT	The document referenced by this method does not exist.
2	eDOCLIST_OUT_RANGE	You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED	Internal error
4	eNOT_DOCUMENTTYPE	You tried to perform an operation upon a parameter that is not a document type.
5	eNOT_LISTTYPE	You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE	You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA	You tried to retrieve data from a document that contained no data.

Example

In this example, the Output structure for Calculate Cost, or the root level document, is accessed with the GetOutputDocs method. The GetNextDoc method gets another Output structure, assuming that there is more than one of them. The Calculate Cost output parameter output_param2_List is a document, so the GetDoc method gets it to the Output structure. output_param2_List is also a document list, so GetNextDoc method gets the next output_param2_List document.

```
Local Interlink &QE_COST;
Local number &count1, &count2;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret1, &ret2;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");

&ret1 = 0;
While (&ret1)
   &ret1 = &CalcCostOut.GetValue("output_param1",&VALUE);
   &ServiceRateDoc = &CalcCostOut.GetDoc("Service_Rate");
   &ret1 = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
   &ret1 = &ServiceRateDoc.GetValue("Rate", &RATE);
   /* Process output values */

   &OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
   &count2 = &CalcCostOut.GetCount("output_param2_List");
   While (&I < &count2)
      &ret2 = &OutlistDoc.GetValue("output_member1", &VALUE1);
      &ret2 = &OutlistDoc.GetValue("output_member2", &VALUE2);
      If &ret2 = 0 Then
         /* Process output values */
         &I = &I + 1;
         &ret2 = &OutlistDoc.GetNextDoc();
      End-If;
   &ret1 = &CalcCostOut.GetNextDoc();
End-While;
```

The following shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example output structure

GetOutputDocs

Syntax

GetOutputDocs("")

Description

The GetOutputDocs method gets the top output document at the root level for a Business Interlink object. This is a hierarchical structure that contains the values for the outputs for this Business Interlink object. The methods that you use to get output values from this document are the hierarchical data methods.

Parameters

A null string.

Returns

An output document. This is the document at the top of the root level of the output document for a Business Interlink object.

Example

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn, &CalcCostOut;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&CalcCostOut = &QE_COST.GetOutputDocs("");

/* You can now execute the Business Interlink object and get the output values. */
```

See Also

Chapter 10, "Business Interlink Class," Using Hierarchical Data (BIDocs), page 306

GetPreviousDoc

Syntax

GetPreviousDoc()

Description

The GetPreviousDoc method gets the previous document from either the root level of the Output structure for a Business Interlink object, or from a document within the Output structure. When these documents are in a list, GetPreviousDoc gets the previous document in the list. You must get the document before you can get its values with GetValue.

Parameters

None.

Returns

The following are the possible returns:

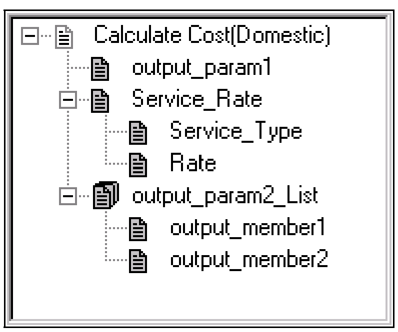
Number	***Enum Value***	***Description***
0	eNoError	The method succeeded
1	eNO_DOCUMENT	The document referenced by this method does not exist.
2	eDOCLIST_OUT_RANGE	You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED	Internal error
4	eNOT_DOCUMENTTYPE	You tried to perform an operation upon a parameter that is not a document type.
5	eNOT_LISTTYPE	You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE	You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA	You tried to retrieve data from a document that contained no data.

Example

In this example, the Output structure for Calculate Cost, or the root level document, is accessed with the GetOutputDocs method. It contains one output parameter, output_param2_List, which is also a list. The GetCount and MoveToDoc methods point to the last output_param2_List document in the list. The GetPreviousDoc method is used in a loop to cycle through the output_param2_List list, starting with the last and ending with the first in the list, and get each output_param2_List document and its values.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");

/* Call GetValue for output_param1, call GetDoc, GetValue for Service_Rate (code⇒
 not shown) */

&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
&count = &CalcCostOut.GetCount("output_param2_List");
&ret = &OutlistDoc.MoveToDoc(&count-1);
&I = &count;
While (&I > 0)
   &ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
   &ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
   If &ret = 0 Then
      /* Process output values */
      &I = &I - 1;
      &retoutput = &OutlistDoc.GetPreviousDoc("");
   End-If;
End-While;
```

The following shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example output structure

GetStatus

Syntax

GetStatus()

Description

The GetStatus method tests the BIDocs document. To find the status for any BIDocs method that does not return a status value, call GetStatus just after you call that BIDocs method.

Parameters

None.

Returns

The following are the possible returns:

Number	*Enum Value*	*Description*
0	NoError	The method succeeded.
1	NO_DOCUMENT	The document referenced by this method does not exist.
2	LIST_OUT_RANGE	You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	DOCUMENT_UNINITIALIZED	Internal error.
4	NOT_DOCUMENTTYPE	You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE	You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE	You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE	You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA	You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR	There was an error with the transaction.

GetValue

Syntax

`GetValue``(`*`paramname`*`,`*`value`*`)`

Description

The GetValue method gets a value from an output parameter within a document of the Output structure for a Business Interlink object.

Parameters

Parameter	***Description***
Paramname	The name of the member of the document that is having a value retrieved from it.
Value	The value that is retrieved. This can be a variable or a record field. The data type can be string, integer, double, float, time, date, or datetime.

Returns

The following are the possible return values:

Return Status for Integer

Number	***Enum value and Meaning***
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

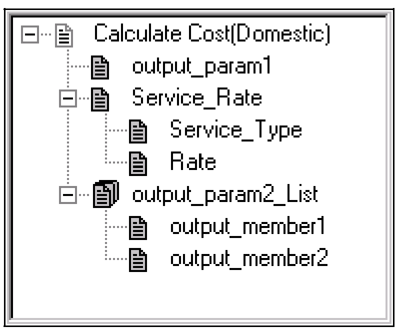
Example

In the following example, the Business Interlink object name is QE_COST.

In the following example, the Output structure for Calculate Cost, or the root level document, is created with the GetOutputDocs method. (If you wanted to create, or get, more Output structures, you would call GetNextDoc.) The Calculate Cost output parameter Service_Rate is a document, so the GetDoc method gets it from the Output structure. Then the GetValue method gets values from each of the Service_Rate document members.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostOut;
Local BIDocs &ServiceRateDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&ret = &CalcCostOut.GetValue("output_param1",&PARAM1);

&ServiceRateDoc = &CalcCostOut.GetDoc("Service_Rate");
&ret = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
&ret = &ServiceRateDoc.GetValue("Rate", &RATE);

/* Call GetDoc, GetValue, GetNextDoc for output_param2_List (code not shown) */
```

The following illustration shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Federal Express plug-in that was modified for this example (output_param1 and output_param2_List were added).



Example output structure

InputRowset

Syntax

`InputRowset``(`*`&Rowset`*`)`

Description

The InputRowset method uses the data in the specified rowset to populate the input buffer, copying *like-named* fields in the appropriate structure. This method assumes that the names of the fields in the rowset match the names of the inputs defined in the Business Interlink definition, and that the structure of both rowsets are the same.

Use this method only if you have a hierarchical data structure and you want to preserve the hierarchy. Otherwise, use BulkExecute or AddInputRow.

Parameters

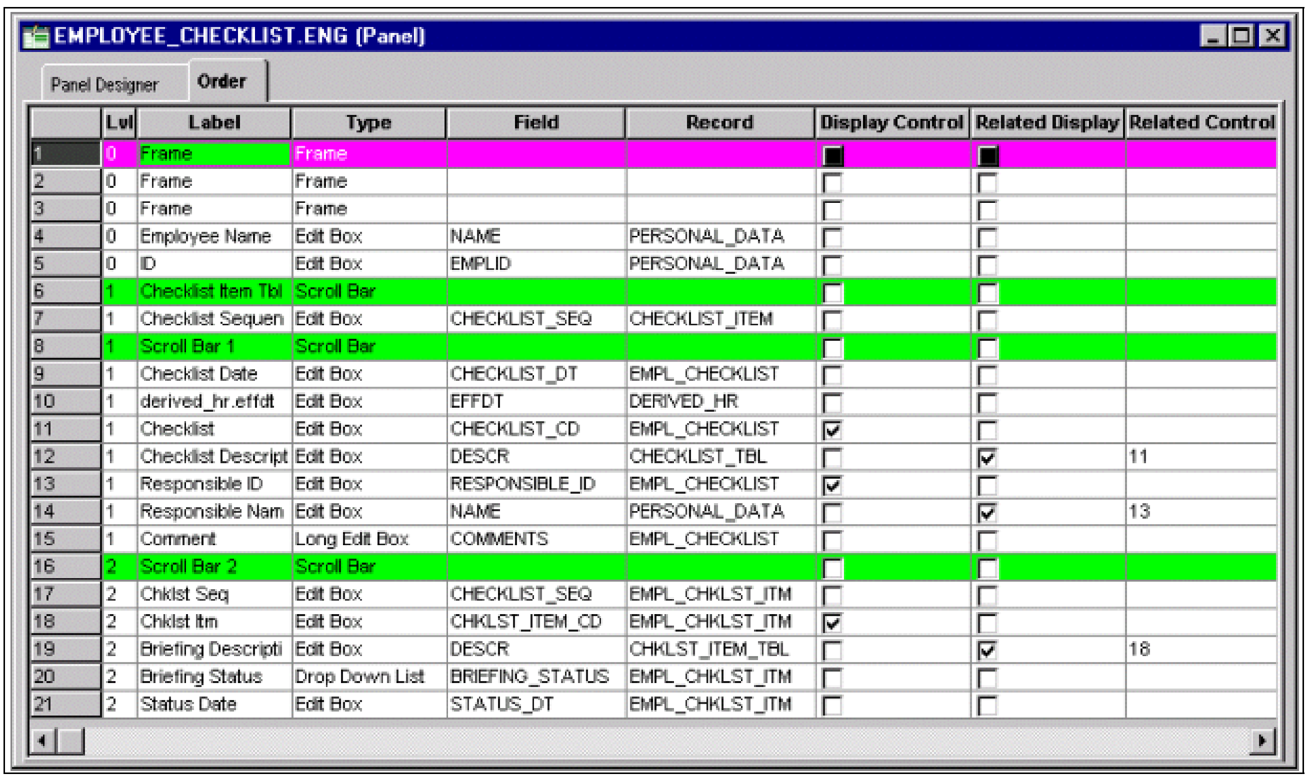
Parameter	*Description*
&Rowset	Specify an existing, instantiated rowset object.

Returns

An optional value: the number of rows inserted into the output buffer.

Example

The example uses the rowset on level one from the EMPLOYEE_CHECKLIST page. A PeopleCode program running in a field on level zero in that panel can access the child rowset (level one), shown below from Scroll Bar 1 to Scroll Bar 2.


EMPLOYEE_CHECKLIST.ENG (Panel)

Panel Designer | Order

	Lvl	Label	Type	Field	Record	Display Control	Related Display	Related Control
1	0	Frame	Frame			■	■	
2	0	Frame	Frame			☐	☐	
3	0	Frame	Frame			☐	☐	
4	0	Employee Name	Edit Box	NAME	PERSONAL_DATA	☐	☐	
5	0	ID	Edit Box	EMPLID	PERSONAL_DATA	☐	☐	
6	1	Checklist Item Tbl	Scroll Bar			☐	☐	
7	1	Checklist Sequen	Edit Box	CHECKLIST_SEQ	CHECKLIST_ITEM	☐	☐	
8	1	Scroll Bar 1	Scroll Bar			☐	☐	
9	1	Checklist Date	Edit Box	CHECKLIST_DT	EMPL_CHECKLIST	☐	☐	
10	1	derived_hr.effdt	Edit Box	EFFDT	DERIVED_HR	☐	☐	
11	1	Checklist	Edit Box	CHECKLIST_CD	EMPL_CHECKLIST	☑	☐	
12	1	Checklist Descript	Edit Box	DESCR	CHECKLIST_TBL	☐	☑	11
13	1	Responsible ID	Edit Box	RESPONSIBLE_ID	EMPL_CHECKLIST	☑	☐	
14	1	Responsible Nam	Edit Box	NAME	PERSONAL_DATA	☐	☑	13
15	1	Comment	Long Edit Box	COMMENTS	EMPL_CHECKLIST	☐	☐	
16	2	Scroll Bar 2	Scroll Bar			☐	☐	
17	2	Chklst Seq	Edit Box	CHECKLIST_SEQ	EMPL_CHKLST_ITM	☐	☐	
18	2	Chklst Itm	Edit Box	CHKLST_ITEM_CD	EMPL_CHKLST_ITM	☑	☐	
19	2	Briefing Descripti	Edit Box	DESCR	CHKLST_ITEM_TBL	☐	☑	18
20	2	Briefing Status	Drop Down List	BRIEFING_STATUS	EMPL_CHKLST_ITM	☐	☐	
21	2	Status Date	Edit Box	STATUS_DT	EMPL_CHKLST_ITM	☐	☐	

EMPLOYEE_CHECKLIST Page structure

EMPL_CHECKLIST is the primary database record for the level one scrollbar on the EMPLOYEE_CHECKLIST page. The following PeopleCode access the level one rowset using EMPL_CHECKLIST. The Business Interlink object name is QE_BI_EMPL_CHECKLIST. This Business Interlink object uses the level one rowset as its input and its output.

The InputRowset method uses this rowset as input for QE_BI_EMPL_CHECKLIST. Then a blank duplicate of the rowset is created with CreateRowset, and then the output of QE_BI_EMPL_CHECKLIST is fetched into the blank rowset with FetchIntoRowset.

```
&MYROWSET = GetRowset(SCROLL.EMPL_CHECKLIST);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.InputRowset(&MYROWSET);
&RSLT = &QE_BI_EMPL_CHECKLIST.Execute();
/* do some error processing */
&WorkRowset = CreateRowset(&MYROWSET);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.FetchIntoRowset(&WorkRowset);

If &ROWCOUNT = 0 Then
/* do some error processing */
Else
   /* Process the rowset from QE_BI_EMPL_CHECKLIST. Check it for errors. */
   For &I = 1 to &WorkRowset.RowCount
      For &K = 1 to &WorkRowset(&I).RecordCount
         &REC = &WorkRowset(&I).GetRecord(&K);
         &REC.ExecuteEdits();
         For &M = 1 to &REC.FieldCount
            If &REC.GetField(&M).EditError Then
            /* there are errors */
            /* do other processing */
            End-If;
         End-For;
      End-For;
   End-for;
End-if;
```

See Also

Chapter 10, "Business Interlink Class," FetchIntoRowset, page 329

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

MoveFirst

Syntax

MoveFirst()

Description

Use the MoveFirst method to support dynamic output. This method moves your cursor to the first column, first row of the output buffer. You must use this method before you try to access any data.

Parameters

None.

Returns

Boolean: True if successfully positioned cursor at the first column, first row of the output buffer, False otherwise.

Example

In the following example, the Business Interlink object name is &MYBI. The example uses MoveFirst to move to the first row of the output buffer, and to the first column, or first field, in that row. The Repeat loop uses MoveNext to go through every row in the output buffer. The For loop processes every field in every row, using the GetFieldCount method to get the number of fields, or outputs, in the row.

```
/* Add inputs to the Business Interlink object, then call Execute to execute the⇒
 Business Interlink object. You are then ready to get the outputs using the⇒
 following code. */

If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            &STRING_VARIABLE = &MYBI.GetFieldValue(&I);
         /* test for and process other field types */
         End-Evaluate;
      End-For;
   Until Not(&MYBI.MoveNext());
Else
   /* Process error - no output buffer */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," GetFieldCount, page 338; Chapter 10, "Business Interlink Class," GetFieldType, page 339; Chapter 10, "Business Interlink Class," GetFieldValue, page 340 and Chapter 10, "Business Interlink Class," MoveNext, page 353

MoveNext

Syntax

MoveNext()

Description

Use the MoveNext method to support dynamic output. This method moves your cursor to the first column of the next row of the output buffer. You can use this method only after you have used the MoveFirst method: otherwise, the system doesn't know where to start.

Parameters

None.

Returns

Boolean: True if successfully positioned cursor at the next row of the output buffer, False otherwise.

Example

In the following example, the Business Interlink object name is &MYBI. The example uses MoveFirst to move to the first row of the output buffer, and to the first column, or first field, in that row. The Repeat loop uses MoveNext to go through every row in the output buffer. The For loop processes every field in every row, using the GetFieldCount method to get the number of fields, or outputs, in the row.

```
/* Add inputs to the Business Interlink object, then call Execute to execute the⇒
 Business Interlink object. You are then ready to get the outputs using the⇒
 following code. */

If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            &STRING_VARIABLE = &MYBI.GetFieldValue(&I);
         /* test for and process other field types */
         End-Evaluate;
      End-For;
   Until Not(&MYBI.MoveNext());
Else
   /* Process error - no output buffer */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," GetFieldCount, page 338; Chapter 10, "Business Interlink Class," GetFieldType, page 339; Chapter 10, "Business Interlink Class," GetFieldValue, page 340 and Chapter 10, "Business Interlink Class," MoveFirst, page 352

MoveToDoc

Syntax

MoveToDoc*(list_number)*

Description

Within a list of documents in the Output structure, the MoveToDoc method moves to the documents given by the parameter *list_number*. After using MoveToDoc, the GetValue method gets the values of the document that is in the *list_number*+1 location in the list.

Parameters

Parameter	*Description*
list_number	The number indicating the document that MoveToDoc moves to. After using MoveToDoc, the GetValue method gets the values of the document that is in the *list_number*+1 location in the list. For example, if *list_number* is zero, then MoveToDoc moves to the first document in the list.

Returns

This method returns an integer. The values are:

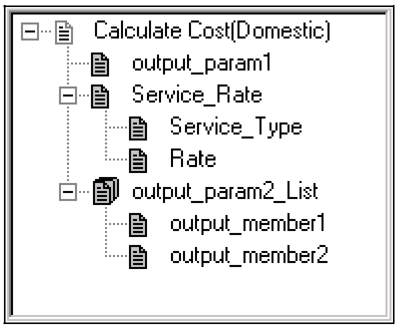
Value	*Description*
0	eNoError. The method succeeded.
1	eNO_DOCUMENT. The document referenced by this method does not exist.
2	eDOCLIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED. Internal error.
4	eNOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	eNOT_LISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE. You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA. You tried to retrieve data from a document that contained no data.

Example

The following example gets the values of the last document in the output_param2_List list. It uses GetCount to get the number of documents in the list, and then uses MoveToDoc to move to the last document in the list.


```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
&count = &CalcCostOut.GetCount("output_param2_List");
&ret = &OutlistDoc.MoveToDoc(&count-1);
&ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
&ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
```

The following illustration shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Federal Express plug-in that was modified for this example (output_param1 and output_param2_List were added).



Example output structure

ResetCursor

Syntax

ResetCursor()

Description

The ResetCursor method resets the cursor in the Output structure for a Business Interlink object to the top. After you call this method, the next time you call GetValue, you get an output value from the first document of the Output structures for a Business Interlink object.

Parameters

None.

Returns

None.

Example

The following code uses ResetCursor to reset the cursor in the Output structure to the top.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostOut;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");

// Perform actions on the output documents (code not shown)

&CalcCostOut.ResetCursor();
```

Incoming Business Interlink Methods

This section describes the PeopleCode methods you use with Incoming Business Interlinks.

AddAttribute

Syntax

AddAttribute*(attributename,attributevalue)*

Description

The AddAttribute method adds an attribute name and its value to an XML element referenced by a BiDocs object.

Parameters

Parameter	***Description***
attributename	String. The name of the attribute.
attributevalue	String. The value of the attribute.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <candidate>
         <user>
            <location scenery="great" density="low" blank="eh?">
            </location>
         </user>
      </candidate>
   </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &candidates, &location, &user;
Local number &ret;

&rootDoc = GetBiDoc("");
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&candidates = &postreqresponse.CreateElement("candidates");
&user = &candidates.CreateElement("user");
&location = &user.CreateElement("location");
&ret = &location.AddAttribute("scenery", "great");
&ret = &location.AddAttribute("density", "low");
&ret = &location.AddAttribute("blank", "eh?");
```

See Also

Chapter 10, "Business Interlink Class," GetStatus, page 347

AddComment

Syntax

AddComment*(comment)*

Description

The AddComment method adds an XML comment after the beginning tag of an XML element referenced by a BiDoc object.

Parameters

Parameter	Description
comment	String. The comment.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <!--this is a comment line-->
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse, &error, &errorcode, &errortext;
Local string &blob;
Local number &ret;
&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&ret = &error.AddComment("this is a comment line");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

See Also

Chapter 10, "Business Interlink Class," GetStatus, page 347

AddProcessInstruction

Syntax

AddProcessInstruction*(instruction)*

Description

The AddProcessInstruction method adds an XML processing instruction to a BiDocs object. Use this method at the root level of the BiDocs object for Incoming Business Interlinks before you add anything else to the BiDocs object.

Parameters

Parameter	***Description***
instruction	String. The processing instruction.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <!--this is a comment line-->
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local number &ret;
&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&ret = &error.AddComment("this is a comment line");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

See Also

Chapter 10, "Business Interlink Class," GetStatus, page 347

AddText

Syntax

AddText*(text)*

Description

The AddText method adds text to an XML element referenced by a BiDocs object.

Parameters

Parameter	*Description*
text	String. The text.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <!--this is a comment line-->
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local number &ret;
&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&ret = &error.AddComment("this is a comment line");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

See Also

Chapter 10, "Business Interlink Class," GetStatus, page 347

CreateElement

Syntax

`CreateElement``(`*`elementname`*`)`

Description

The CreateElement method creates an XML element with the given name within a BiDoc object.

Parameters

Parameter	***Description***
elementname	String. The XML element name.

Returns

BiDocs. The reference to the created element.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopleCode that builds it.


```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local number &ret;

&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

GenXMLString

Syntax

GenXMLString()

Description

The GenXMLString method creates an XML string from a BiDocs object. The BiDocs object must contain the shape and data needed for an XML string. This is part of the Incoming Business Interlink functionality, which enables PeopleCode to receive an XML request and return an XML response.

Parameters

None.

Returns

String. The XML string containing the shape and data of the BiDocs object. For example, you can use this method to create an XML string containing an XML response.

Example

The following example takes a BiDocs structure that contains an XML response and puts that into a text string. After this is done, the %Response.Write function can send this as an XML response.

```
Local BIDocs &rootDoc;
Local string &xmlString;

/* Create a BiDoc structure containing the data and shape of your XML response ⇒
(code not shown) */

/* Generate the XML string containing the data and shape of your XML response from⇒
 the BiDoc structure */
&xmlString = &rootDoc.GenXMLString();
%Response.Write(&xmlString);
```

GetAttributeName

Syntax

GetAttributeName*(attributenumber)*

Description

The GetAttributeName method gets the name of an attribute within an XML element referenced by a BiDocs object.

Parameters

Parameter	*Description*
attributenumber	Number. The index number of the attribute.

Returns

String. The name of the attribute.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <location scenery="great" density="low" blank="eh?">
         <city>San Rafael</city>
         <state>CA</state>
         <zip>94522</zip>
         <country>US</country>
      </location>
   </postreq>
```

Here is the PeopleCode that gets the name of the second attribute in the location node. &attrName is density.

```
Local BIDocs &rootInDoc, postreqDoc, &locationDoc;
Local string &blob, &attrName;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&locationDoc = &postreqDoc.GetNode("location");
&attrName = &locationDoc.GetAttributeName(2);
```


GetAttributeValue

Syntax

GetAttributeValue({*attributename* | *attributenumber*})

Description

The GetAttributeValue method gets the value of an attribute within an XML element referenced by a BiDocs object.

Parameters

Parameter	*Description*
attributenumber | *attributename*	Specify the attribute that you want to get the value for. You can specify either the attribute number (1 for the first attribute, 2 for the second, and so on) or the attribute name (an XML tag.)

Returns

String. The value of the attribute.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <location scenery="great" density="low" blank="eh?">
         <city>San Rafael</city>
         <state>CA</state>
         <zip>94522</zip>
         <country>US</country>
      </location>
   </postreq>
```

Here is the PeopleCode that gets the value of the second attribute in the location node. &attrValue is low.

```
Local BIDocs &rootInDoc, &postreqDoc, &locationDoc;
Local string &blob, &attrValue;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&locationDoc = &postreqDoc.GetNode("location");
&attrValue = &locationDoc.GetAttributeValue(2);
```


GetNode

Syntax

```
GetNode({nodename | nodenumber})
```

Description

The GetNode method returns a BiDocs reference to a child XML node (element or comment). Use the GetNode method upon a BiDocs reference to an XML element to access the child nodes for that element.

Parameters

Parameter	*Description*
nodenumber | nodename	Specify the node that you want to reference. You can specify either a node number (the first node is 1, the second 2, and so on) or a node name (that is, the XML tag.)

Returns

BiDocs. The returned XML element within a BiDocs object.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets the postreqDoc element and the projtitle element.

```
Local BIDocs &rootInDoc, &postreqDoc, &projtitleDoc;
Local string &name, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&projtitleDoc = &postreqDoc.GetNode("projtitle");
```

Here is the PeopleCode that gets the postreqDoc element, the projtitle element, and the third projsubtitle element.


```
Local BIDocs &rootInDoc, &postreqDoc, &projtitleDoc, &projsubtitleDoc;
Local string &name, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode(1);
&projtitleDoc = &postreqDoc.GetNode(2);
&projsubtitleDoc = &projtitleDoc.GetNode(3);
```

To parse a list of elements like <projsubtitle>, where elements have the same name, you must call GetNode using an index number rather than the element name.

ParseXMLString

Syntax

ParseXMLString(*XMLstring* [, *DTDValidation*])

Description

The ParseXMLString methods fills an existing BiDocs object with the data and shape from an XML string. This is part of the Incoming Business Interlink functionality, which enables PeopleCode to receive an XML request and return an XML response.

Parameters

Parameter	***Description***
XMLstring	A string containing an XML document.
DTDValidation	Specify whether to validate a document type definition (DTD.) This parameter takes a Boolean value. If you specify true, the DTD validation occurs if a DTD is provided. If you specify false, and if a DTD is provided, it is ignored and the XML isn't validated against the DTD. The default value for this parameter is false. In the case of application messaging, if a DTD is provided, it's always ignored and the XML isn't validated against the DTD. If the XML cannot be validated against a DTD, an error is thrown saying that there was an XML parse error.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

Example

The following example gets an XML request, creates an empty BiDoc, and then fills the BiDoc with the data and shape contained in the XML string. After this is done, the GetDoc method and the GetValue method can get the value of the skills XML element, which is contained within the postreq element in the XML request.


```
Local BIDocs &rootInDoc, &postreqDoc;
Local string &blob;
Local number &ret;

&blob = %Request.GetContentBody();
/* process the incoming xml(request)- Create a BiDoc and fill with the request*/
&rootInDoc = GetBiDoc("");
&ret = &rootInDoc.ParseXMLString(&blob);
&postreqDoc = &rootInDoc.GetDoc("postreq");
&ret = &postreqDoc.GetValue("skills", &skills);
```

See Also

Chapter 10, "Business Interlink Class," GetStatus, page 347

Incoming Business Interlink Properties

This section describes the PeopleCode properties you use with Incoming Business Interlinks.

AttributeCount

Description

The AttributeCount property gets the number of attributes within an XML element referenced by a BiDocs object.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <location scenery="great" density="low" blank="eh?">
         <city>San Rafael</city>
         <state>CA</state>
         <zip>94522</zip>
         <country>US</country>
      </location>
   </postreq>
```

Here is the PeopleCode that gets the number of attributes in the location XML element. &count should be 3, for scenery, density, and blank.


```
Local BIDocs &rootInDoc, &postreqDoc, &locationDoc;
Local string &blob;
Local number &count;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&locationDoc = &postreqDoc.GetNode("location");
&count = &locationDoc.AttributeCount;
```

ChildNodeCount

Description

The ChildNodeCount property returns the number of XML child nodes within the element referenced by the BiDocs object. Child nodes include XML elements, comments, and processing instructions.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <!--this is a comment line-->
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the XML code that gets the number of nodes within <postreq> and <projtitle>. &count1 is 2, for <email> and <projtitle>, and &count2 is 4, for the three <projsubtitle> nodes and the comment node.

```
Local BIDocs &rootInDoc, &projtitleDoc;
Local string &blob;
Local number &count1, &count2;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&count1 = &postreqDoc.ChildNodeCount;
&projtitleDoc = &postreqDoc.GetNode("projtitle");
&count2 = &projtitleDoc.ChildNodeCount;
```


NodeName

Description

The NodeName property gets the name of an XML element referenced by a BiDocs object. Use this to get the name of an XML element when you used GetNode with an index number to retrieve it (meaning that you did not have the name of the XML element when you used GetNode).

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets the name of the <email> element, email.

```
Local BIDocs &rootInDoc, &postreqDoc, &emailDoc;
Local string &emailName, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode(1);
&emailDoc = &postreqDoc.GetNode(1);
&emailName = &emailDoc.NodeName;
```

NodeType

Description

The NodeType property returns the type of an XML tag within a BiDocs object as an integer. The values are:

Value	***Description***
1	Element (a normal XML tag)
7	Processing instruction
8	Comment

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <!--this is a comment-->
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets types: &xmlprocType is 7 for processing instruction, postreqDoc is 1 for element, and commentType is 8 for comment.

```
Local BIDocs &rootInDoc, &postreqDoc, &commentDoc;
Local number &xmlprocType, &postreqType, &commentDoc;
Local string &blob;
&blob = %Request.GetContentBody();

/* <?xml version="1.0"?> */
&rootInDoc = GetBiDoc(&blob);
&xmlprocType = &rootInDoc.NodeType;

/* <postreq> */
&postreqDoc = &rootInDoc.GetNode(1);
&postreqType = &postreqDoc.NodeType;

/* <!--this is a comment--> */
&commentDoc = &postreqDoc.GetNode(2);
&commentType = &commentDoc.NodeType;
```

NodeValue

Description

The NodeValue property returns the value of a node within an XML document as a string.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets the value of the third <projsubtitle> element, third_subtitle.

```
Local BIDocs &rootInDoc, &postreqDoc, &projtitleDoc, &projsubtitleDoc;
Local string &name, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode(1);
&projtitleDoc = &postreqDoc.GetNode(2);
&projsubtitleDoc = &projtitleDoc.GetNode(3);
&projsubtitleName = &projsubtitleDoc.NodeValue;
```

Business Interlink Class Property

This section explains the StopAtError property.

StopAtError

Description

The StopAtError property specifies whether the execution of the PeopleCode program stops when there's an error, or if the PeopleCode program tries to capture the errors.

This property takes a Boolean value: True to halt execution of your PeopleCode program at an error, False to continue executing. The default value is True.

This property is read-write.

Example

```
&QE_RP_SRAALL_1.StopAtError = False;
```

Configuration Parameters

There are two types of configuration parameters: the ones defined by the Interlink plug-in the Business Interlink definition is based on, and the ones that are standard, that is, defined for every Business Interlink object.

All configuration parameters must be set before you add any data to the input buffers.

In this section, we discuss the following parameters:

- Interlink plug-in configuration parameters
- URL configuration parameter
- BIDocValidate configuration parameter

Interlink Plug-in Configuration Parameters

The configuration parameters defined by the Interlink plug-in are accessed as if they were properties on the Business Interlink object. That is, in PeopleCode, you assign the value of a configuration parameter by using the Business Interlink object followed by a dot and the configuration parameter, in this format:

```
&MYINTERLINK.parameter = value;
```

You can also return the value of a configuration parameter:

```
&MYVALUE = &MYINTERLINK.parameter;
```

Each Business Interlink plug-in has its own set of configuration parameters. For example, the email project uses configuration parameters of SMTP_MAIL_SERVER, POP3MAIL_SERVER, LOGIN_NAME, PASSWORD, SENDERS_EMAIL_ADDRESS and REPLY_EMAIL_ADDRESS, while the Red Pepper transaction driver uses SERVER_NAME, RSERVER_HOST, RSERVER_PORT, and TIMEOUT.

You can specify default values for every configuration parameter in the Business Interlink definition (created in Application Designer.) These values are used if you create a PeopleCode "template" by dragging the Business Interlink definition from the Project window in Application Designer to an open PeopleCode editor window.

In the following example, the Interlink object name is QE_RP_SRAALL_1, and the driver is the Red Pepper driver:

```
&QE_RP_SRAALL_1.SERVER_NAME = "server";
&QE_RP_SRAALL_1.RSERVER_HOST = "pt-sun02.peoplesoft.com";
&QE_RP_SRAALL_1.RSERVER_PORT = "9884";
&QE_RP_SRAALL_1.TIMEOUT = 999;
&QE_RP_SRAALL_1.URL = "HTTP://www.PeopleSoft.com";
&QE_RP_SRAALL_1.StopAtError = False;
```

URL Configuration Parameter

Specifies the location and name of the Business Interlink plug-in to be used to connect to the external system. This configuration parameter takes a string value.

If the plug-in is located on a web server, you have to specify the name of the web server.

If you specify a file, you can specify either a relative or an absolute path:

- If you specify an absolute path, you must specify the drive letter:

  ```
  &MYBI.URL = "File://D:\TEMP.MYDLL";
  ```

- If you specify a relative path, just use the file name:

  ```
  &MYBI.URL = "File://TEMP.MYDLL";
  ```


If you specify a relative path, the system firsts looks for the file in the Location directory (specified by the user when the Business Interlink was first created), then it looks in the directory where PeopleTools is installed, in the PSTOOLS/Interface Drivers directory.

BIDocValidate Configuration Parameter

Specifies whether the system should verify whether the hierarchical data object (BIDoc) exists before adding or getting values from it. This configuration parameter takes a Boolean value: True if the system should verify before accessing the object, False otherwise.

The default value is True.

If this configuration parameter is specified as True, and the object specified doesn't exist, the PeopleCode program halts execution and an error is displayed.

Chapter 11

Charting Classes

This chapter provides an overview of the PeopleSoft charting classes and discusses:

- Creating PeopleSoft charts.
- Using the Chart class.
- Using the Gantt class.
- Using the OrgChart class.
- Using the RatingBoxChart class.
- Using the Charting Classes Reference.
- Charting classes built-in functions.

Understanding the Charting Classes

PeopleTools delivers four charting classes. One general charting class supports a range of standard chart types, and three specialized charting classes address specific charting needs. Each charting class is discussed in detail in following sections.

The charting classes are:

- Chart

  Use the Chart class to visually display data series in common formats, including bar charts, line charts, pie charts, and bubble charts.

- Gantt

  Use the Gantt class to create interactive Gantt charts that enable you to display and edit project and task information.

- OrgChart

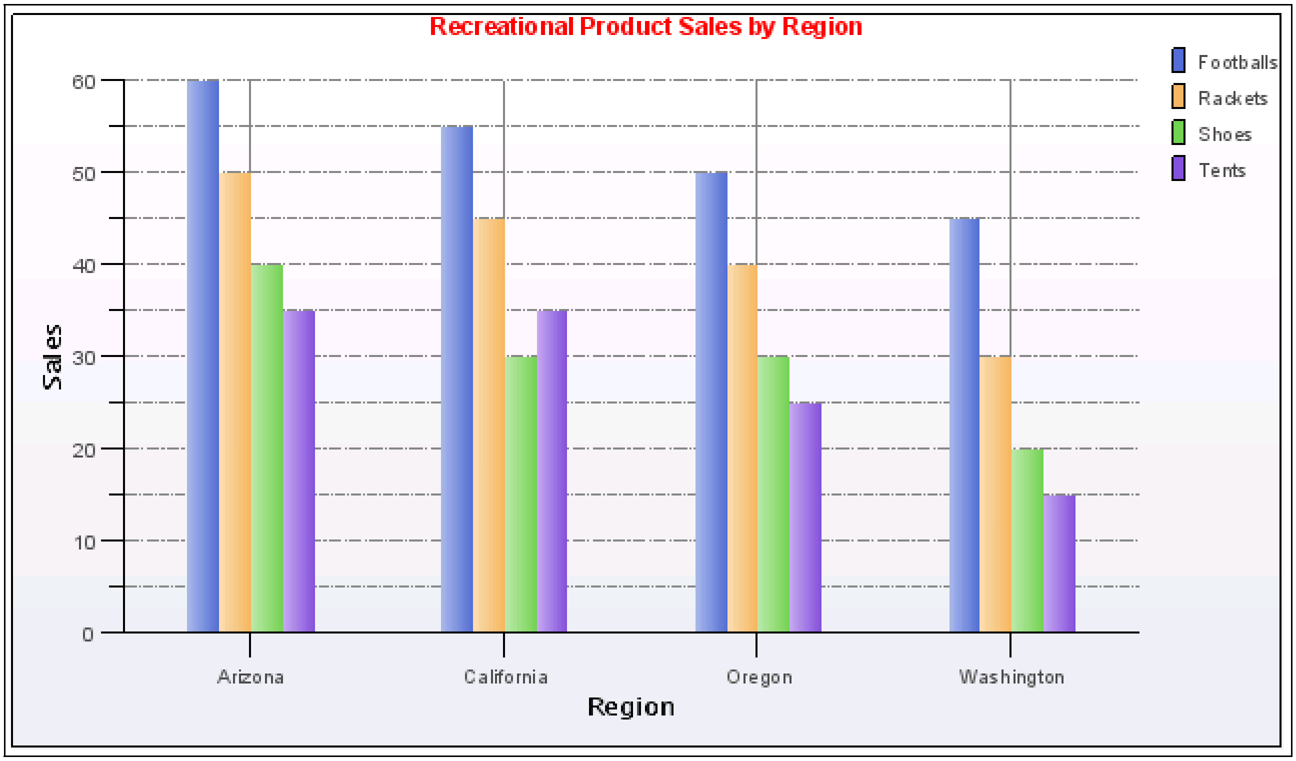
  Use the OrgChart class to create interactive organization charts that enable you to visually represent a hierarchy of information as a series of connected nodes.

- RatingBoxChart

  Use the RatingBoxChart class to create rating box charts that enable you to display and manipulate points of information in two-dimensional bins.

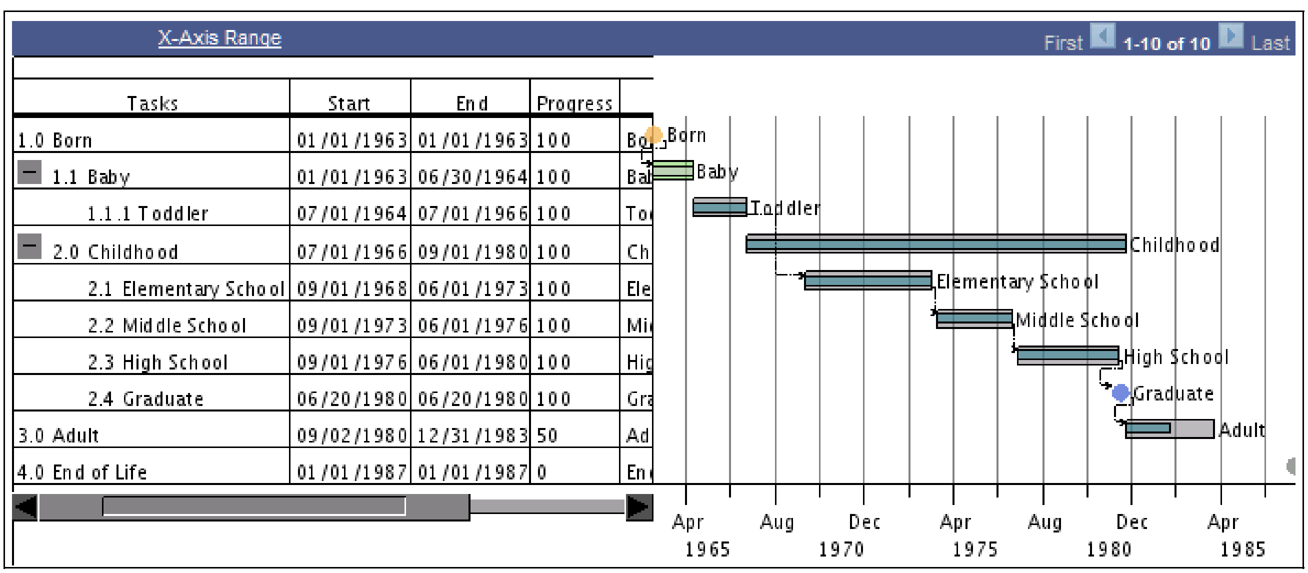
The following examples show each of the four chart classes:

This chart was created from the Chart class with a chart type of 2D Bar:



Two-dimensional bar chart

This example shows a Gantt chart:

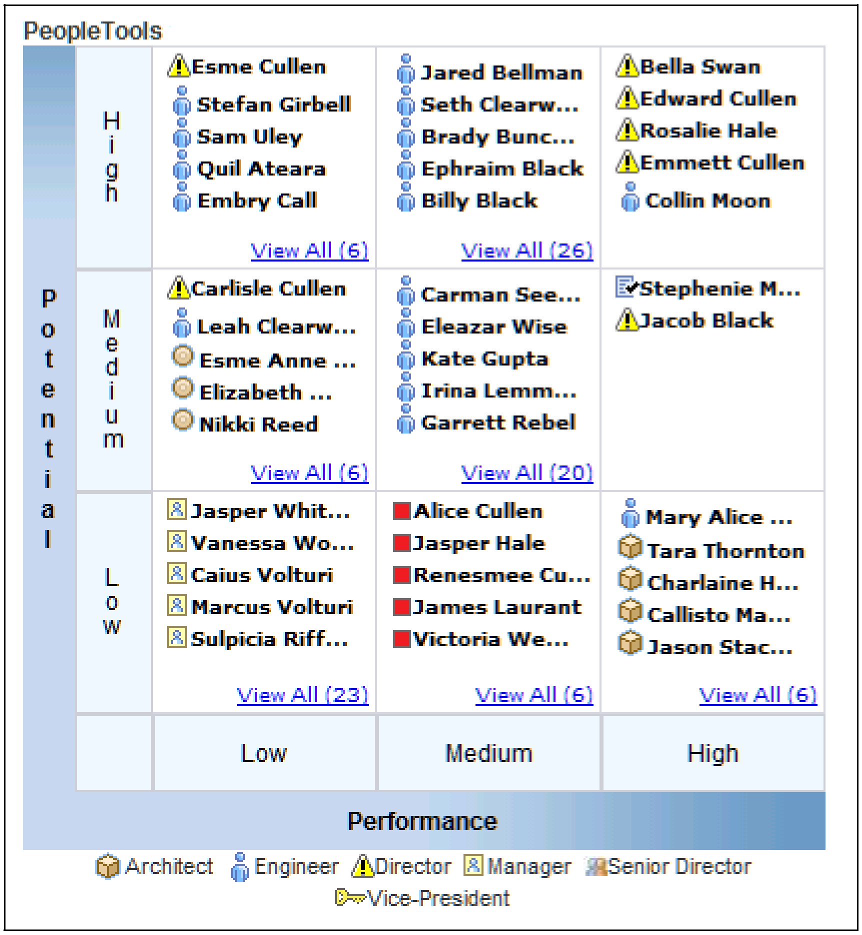


Gantt chart

This example shows an organization chart:

Organization chart

This example shows a rating box chart:



Rating box chart

See Also

Chapter 11, "Charting Classes," Creating PeopleSoft Charts, page 378

Chapter 11, "Charting Classes," Using the Chart Class, page 394

Chapter 11, "Charting Classes," Using the Gantt Class, page 399

Chapter 11, "Charting Classes," Using the OrgChart Class, page 407

Chapter 11, "Charting Classes," Using the RatingBoxChart Class, page 425

Creating PeopleSoft Charts

This section discusses the process of creating charts using any of the PeopleSoft charting classes. Further details can be found in the specific section for each chart class.

These topics relate to all of the charting classes:

- Creating a chart on a page

- Creating a chart using an iScript
- Font considerations
- Component Processor considerations
- Translation considerations
- Label considerations
- Style sheets
- Using style sheets with RevertToPre850
- Using style sheets with PeopleTools Version 8.50 charts
- Chart colors

Creating a Chart on a Page

This section summarizes the basic steps to create a chart. The basic steps are essentially the same for all four charting classes.

At the end of this chapter are examples of charts with step-by-step instructions and complete PeopleCode programs.

The Chart class section has fundamental, detailed instructions for creating a chart using the Chart class. Each specialized class – Gantt class, OrgChart class, and RatingBoxChart class – has detailed instructions that build on the Chart class instructions.

See Chapter 11, "Charting Classes," Creating a Chart Using the Chart Class, page 557; Chapter 11, "Charting Classes," Using the Chart Class, page 394; Chapter 11, "Charting Classes," Using the Gantt Class, page 399; Chapter 11, "Charting Classes," Using the OrgChart Class, page 407 and Chapter 11, "Charting Classes," Using the RatingBoxChart Class, page 425.

Note. The following steps are for creating a chart on a PeopleSoft Pure Internet Architecture page. You can also create a chart for a web page using an iScript. Only Chart class charts or Gantt class charts can be created using an iScript.

See Chapter 11, "Charting Classes," Creating a Chart Using an iScript, page 596.

To create a chart on a page:

1. In Application Designer, place a chart control on a page and associate the chart with a record field.
2. Create a record to hold your chart data.

   For charts made using the OrgChart class and the RatingBoxChart class, you also create a record that controls the appearance of the chart. Place each record on a level-1 grid on the page so the chart records are present in the component buffer at runtime. You can optionally hide the grid. You can also optionally place the grid on another page on the same component.

   Place the page on a component and register the component.

3. Add PeopleCode, probably on the page Activate event, to instantiate the chart object and associate it with the page control.

   For example, this is the minimum PeopleCode to create a bar chart using the Chart class.

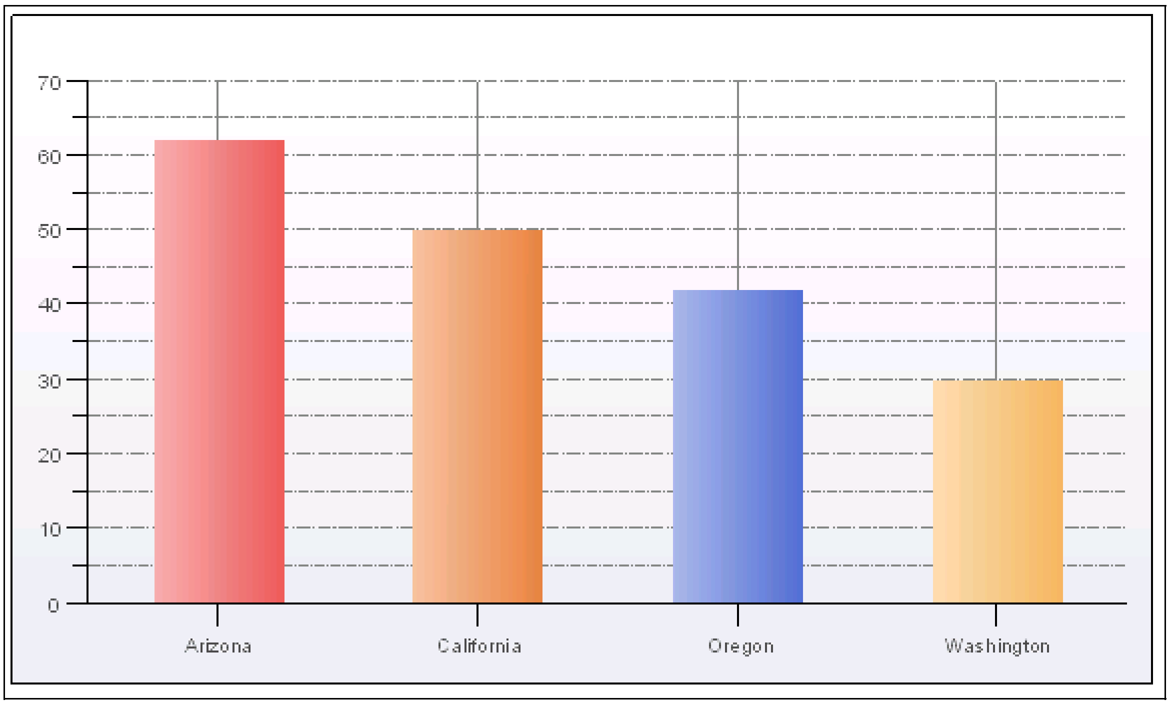
```
/* Declare a chart object */

Component Chart &cChart;

/* Instantiate the chart object and associate the chart
** with the chart page control */

&cChart = GetChart(SC_CHART_DUMREC.QE_CHART_FIELD);

/* Specify the chart data record and specify the record
**  fields for the X-Axis data and Y-Axis data*/

&cChart.SetData(Record.SC_CHART_RECORD);
&cChart.SetDataXAxis(SC_CHART_RECORD.SC_PRODUCT);
&cChart.SetDataYAxis(SC_CHART_RECORD.SC_SALES);
```


4. View the chart in the browser.

   Suppose you have this data in SC_CHART_RECORD:

Region	***Sales***
Arizona	62
California	50
Oregon	42
Washington	30

This is the bar chart generated by the example code and data. The default chart type for the Chart class is a bar chart. Since no colors are specified, the chart uses the default colors.



Default Chart class bar chart

5. Add PeopleCode to control the behavior of the chart or to modify its appearance at runtime. For example, for a bar chart you might add FieldChange PeopleCode to the Y-axis field on your data record to display information when a user clicks a data point.

   See Chapter 11, "Charting Classes," Adding Drilldown, page 584.

   You can also add FieldChange PeopleCode to a Gantt chart, organization chart, or rating box chart to enable the user to interact with the chart.

See Also

Chapter 11, "Charting Classes," Creating a Chart Using the Chart Class, page 557

Chapter 11, "Charting Classes," Using the Chart Class, page 394

Chapter 11, "Charting Classes," Using the Gantt Class, page 399

Chapter 11, "Charting Classes," Using the OrgChart Class, page 407

Chapter 11, "Charting Classes," Using the RatingBoxChart Class, page 425

Creating a Chart Using an iScript

If possible, you should put a chart on a page definition. If not possible, then you can build a chart at runtime using an iScript and call the chart using a URL. Use the CreateObject function in the iScript instead of the GetChart function. You can then use the GetChartURL Response class method to use the URL in your application.

Note. You can create only Chart class and Gantt class charts using an iScript. OrgChart class and RatingBoxChart class charts cannot be created with an iScript.

For example, this PeopleCode would build a chart using a standalone rowset:

```
Function IScript_GetChartURL()
local object &MYCHART;
local string &MYURL;

&MYCHART = CreateObject("Chart");
&MYCHART.SetData(xx);

/* xx will be a data row set */

&MYURL = %Response.GetChartURL(&MYCHART);

/* use &MYURL in your application */
...
End-Function;
```

See Also

Chapter 11, "Charting Classes," Creating a Chart Using an iScript, page 596

Font Considerations

For Chart class charts and Gantt class charts, the environment variable JAVA_FONTS must be set correctly so the fonts that are used in charting can be picked up by the application server's Java Virtual Machine. It must be set to include the paths to any TrueType fonts that are used to support extended languages. You can set this variable at the system level or in psconfig.sh.

The following is an example of this variable being set:


```
JAVA_FONTS=$PS_HOME/jre/lib/fonts:/usr/share/fonts/default/TrueType:/usr/share
/fonts/ja/TrueType:/usr/share/fonts/zh_CN/TrueType:/usr/share/fonts/zh_TW/TrueType
:/usr/share/fonts/ko/TrueType;export JAVA_FONTS
```

Component Processor Considerations

Sometimes, records at level zero on a page are considered *work* records, even when they are not specified as such in PeopleSoft Application Designer. Work records are not updated by the database and are skipped when the component is run.

A level-zero record is marked as a work record when any of these conditions is true:

- The record is designated as a work record in Application Designer.
- All fields for the record are used as read-only fields.
- All values for these fields can be read from the input keys.

When a chart field is attached to a level-zero record field of a record considered to be a work record, it is also skipped when the system determines that record's 'work' status.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Referencing Data in the Component Buffer"

Translation Considerations

If you hard code a value for a label, a title, a data hint, and so on, then that value is *not* translated. Be sure to specify field values as message catalog entries that will be translated.

Label Considerations

The appearance of Chart class chart labels is based on the size of the axis, fonts, and charts on the page. If all the labels for your chart do not appear, then you should either make the chart larger or the font smaller.

Pie chart labels are staggered, so that, in most cases, both labels can be read. However, if one or both of the labels are very long, they could be truncated or completely missing from the chart.

For Gantt charts, use double quotes to correctly display the use of single quotes in your label, if applicable.

Style Sheets

A style sheet is a definition, like a record definition or field definition, that you create in PeopleSoft Application Designer. It contains a collection of formatting styles, called style classes, each of which can be applied to either the entire chart or to specific elements within a chart.

The following style sheet properties are supported for PeopleSoft charting:

- Font family

- Font foreground color
- Font size
- Font style (only Normal and Italic are supported)
- Font weight (only Normal and Bold are supported)

You can use a style sheet, and the style classes in a style sheet, for the different parts of a chart. You can specify a style sheet for the entire chart (using either the chart Style or StyleSheet property) and for an element (using that element's style property.)

Using Style Sheets with RevertToPre850

How your chart uses style sheets is very different depending on whether you want your chart to use the PeopleTools Version 8.50 appearance or the pre-8.50 appearance.

The overall look of charts changed with the introduction of new style sheets and colors in PeopleTools Version 8.50. The way style sheets are applied to charts changed at the same time.

If you want your chart to use the pre-8.50 look, set the RevertToPre850 Chart class or Gantt class property to *True.* If the RevertToPre850 property is not set or is set to *False,* then your chart will use the PeopleTools Version 8.50 look. By default, the property is *False.*

Warning! By default, all charts use the new 8.50 chart style classes, including charts that were created before PeopleTools Version 8.50.

Any custom Chart class properties or Gantt class properties that were set in PeopleCode for existing charts that use the new 8.50 chart style classes will be ignored if those properties are defined in the 8.50 chart style classes.

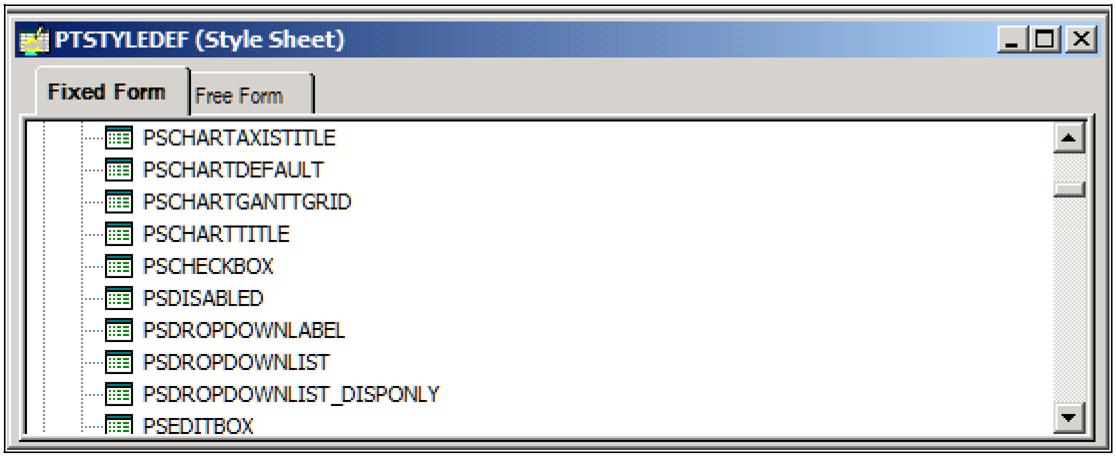
A new property, RevertToPre850, enables you to maintain the pre-8.50 visual style of an existing chart. If you want an existing chart to use pre-8.50 style classes, you must set the chart property RevertToPre850 to *True.*

RevertToPre850 is not available for charts created using the OrgChart class or the RatingBox class. These chart classes were introduced in PeopleTools Version 8.50.

See Chapter 11, "Charting Classes," RevertToPre850, page 463 and *PeopleTools 8.51 PeopleBook: System and Server Administration*, "Using PeopleTools Utilities," PeopleTools Options.

PTSTYLEDEF Style Sheet

The default style sheet associated with a Chart class chart or Gantt class chart with RevertToPre850 set to True is PTSTYLEDEF.



PTSTYLEDEF style sheet showing the chart style sheets

The chart style classes associated with the PTSTYLEDEF style sheet include:

Chart Element	*Style Class Name*
Style XAxisStyle YAxisStyle LegendStyle Also controls the chart section of the Gantt chart	PSCHARTDEFAULT
MainTitleStyle	PSCHARTTITLE
XAxisTitleStyle YAxisTitleStyle	PSCHARTAXISTITLE
Controls the table section of the Gantt chart	PSCHARTGANTTGRID

RevertToPre850 Override Rules

The application style sheet, which is specified on the PeopleTools Options page, controls the formatting of all charts in an application unless it is overridden.

The override rules are:

1. If you specify the style property for any element (XAxisStyle, XAxisTitleStyle, YAxisStyle, YAxisTitleStyle, LegendStyle, MainTitleStyle, and so on), then the style is used.

2. If you do not specify the style property for any element of a chart object but you specify the StyleSheet property for the chart object itself, then that style sheet is used for the unspecified elements.

3. If you do not specify styles for any elements of a chart object and you do not specify a style sheet for the chart object itself, then a default style sheet is used.

Note. Oracle recommends not using your own style classes or style sheets unless absolutely necessary. If you modify styles you must test changes in all supported browsers. Some changes can have different effects in different browsers.

See Also

Chapter 11, "Charting Classes," Using Style Sheets with Gantt Charts, page 406

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Style Sheet Definitions"

Chapter 11, "Charting Classes," Considerations for Gantt Charts, page 391

Using Style Sheets with PeopleTools Version 8.50 Charts

This section describes how to use style sheets if you have not set the RevertToPre850 property or have set the property to False.

The application style sheet, which is specified on the PeopleTools Options page, controls the formatting of all charts in an application unless it is overridden.

The override rules are:

1. If a style sheet is specified for the Chart object or Gantt object using the StyleSheet property, then use this style sheet. (The StyleSheet property only applies to Chart class and Gantt class charts. OrgChart class and RatingBoxChart class charts do not use the StyleSheet property.)

2. If no style sheet is specified for the chart object but the style sheet is specified for the page in Application Designer, then use the style sheet on the page as the style sheet of the chart.

3. If no page style sheet is specified, then use the style sheet specified on the PeopleTools Options page.

4. If no style sheet is specified on the PeopleTools Options page, use the default style sheet, PTSTYLEDEF.

Modifying Style Classes

You modify the appearance of a chart by modifying the style classes that control the appearance. The steps you follow to modify styles depends on the extent of your modification. You can, for instance, modify every style in every chart in your application, or you can modify just one style for a single chart.

The following table summarizes the changes to style sheets and style classes depending on the extent of the modification you intend to make.

Note. Do not change the delivered style sheets. Oracle recommends that you use the default styles wherever possible.

Extent of Modification	***Steps***
All charts in the application	1. Clone a style sheet, probably the default style sheet. For example, in Application Designer open the style sheet PTSTYLEDEF and save it as MySTYLEDEF. 2. Modify the style classes in the new style sheet. 3. Specify the new style sheet on the PeopleTools Options page. See *PeopleTools 8.51 PeopleBook: System and Server Administration*, "Using PeopleTools Utilities," PeopleTools Options.
All charts on a page (This method only applies to PeopleTools Version 8.50 charts.)	1. Clone a style sheet, probably the default style sheet. 2. Modify the style classes in the new style sheet. 3. Specify the new style sheet on the chart page using the page properties dialog. See *PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide*, "Creating Style Sheet Definitions," Overriding the Default Page Style Sheet.
One chart (This method uses the StyleSheet property, which is only used by the Chart class and the Gantt class.)	1. Clone a style sheet, probably the default style sheet. 2. Modify the style classes in the new style sheet. 3. Specify the new style sheet using the Chart class or Gantt class StyleSheet property. See Chapter 11, "Charting Classes," StyleSheet, page 464.

<table>
<tr><th>Extent of Modification</th><th>Steps</th></tr>
<tr><td>One style attribute on one chart</td><td>
1. Clone a style sheet, probably the default style sheet.
2. Copy and paste and rename a style class.

For example, copy the style class PSCHARTTITLE. Paste and name the new class MYCHARTTITLE.
3. Modify the new style class.
4. Specify the new style sheet on the chart page using the page properties dialog.

or

Specify the new style sheet using the Chart class or Gantt class StyleSheet property (Chart class or Gantt class only).

For example:

<code>&cChart.StyleSheet = STYLESHEET.&MyStyleDef;</code>
5. Assign the new style class to the corresponding style property.

For example:

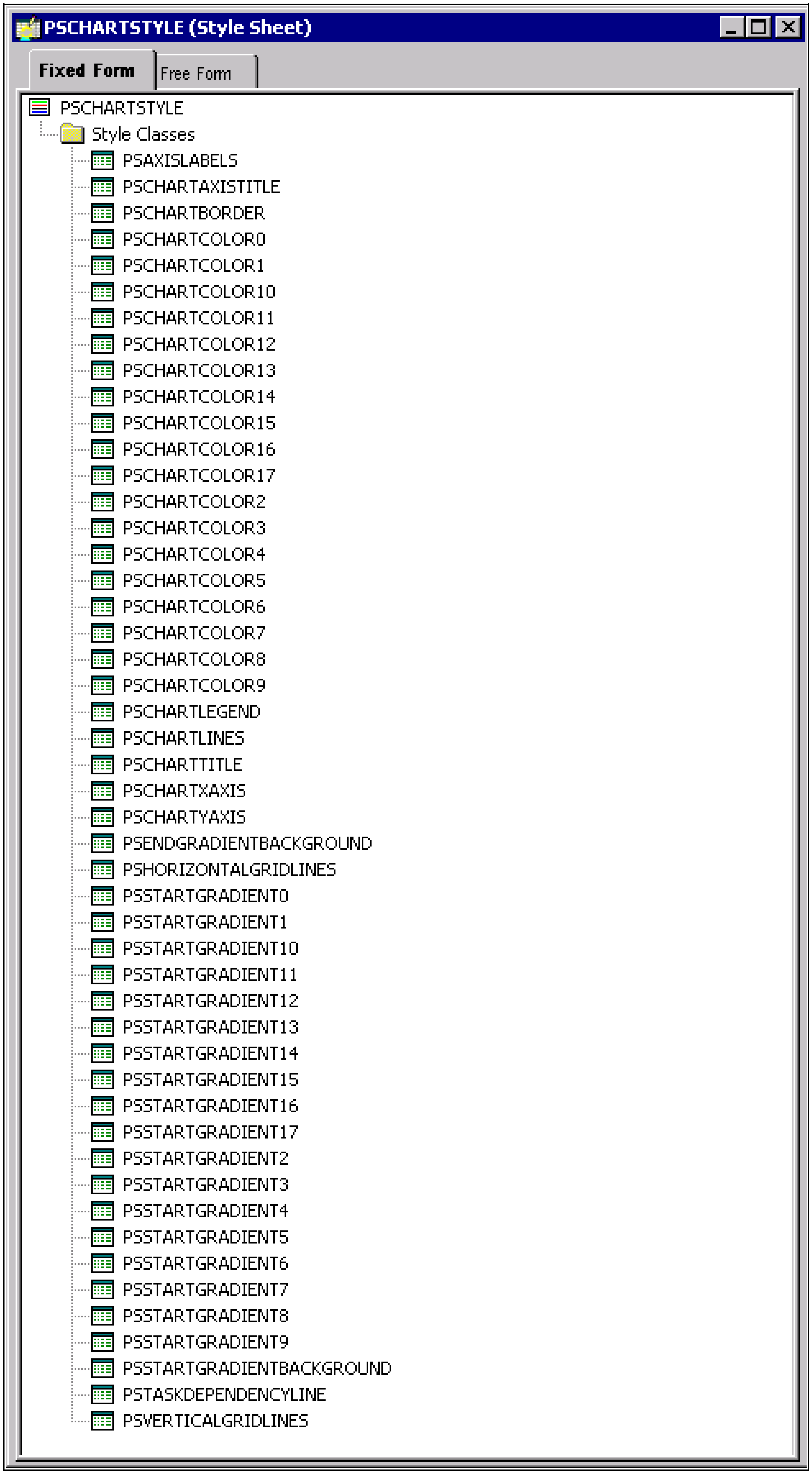
<code>&cChart.&cMyChart.MainTitleStyle = STYLESHEET.MYChartTitle;</code>
</td></tr>
</table>

Note. Oracle recommends not using your own styles or style sheets unless absolutely necessary.

See *PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide*, "Creating Style Sheet Definitions" and *PeopleTools 8.51 PeopleBook: System and Server Administration*, "Using PeopleTools Utilities," PeopleTools Options.

PSCHARTSTYLE

The style classes in PSCHARTSTYLE are used by the Chart class and the Gantt class.


PSCHARTSTYLE (Style Sheet)
Fixed Form
Free Form
PSCHARTSTYLE
Style Classes
PSAXISLABELS
PSCHARTAXISTITLE
PSCHARTBORDER
PSCHARTCOLOR0
PSCHARTCOLOR1
PSCHARTCOLOR10
PSCHARTCOLOR11
PSCHARTCOLOR12
PSCHARTCOLOR13
PSCHARTCOLOR14
PSCHARTCOLOR15
PSCHARTCOLOR16
PSCHARTCOLOR17
PSCHARTCOLOR2
PSCHARTCOLOR3
PSCHARTCOLOR4
PSCHARTCOLOR5
PSCHARTCOLOR6
PSCHARTCOLOR7
PSCHARTCOLOR8
PSCHARTCOLOR9
PSCHARTLEGEND
PSCHARTLINES
PSCHARTTITLE
PSCHARTXAXIS
PSCHARTYAXIS
PSENDGRADIENTBACKGROUND
PSHORIZONTALGRIDLINES
PSSTARTGRADIENT0
PSSTARTGRADIENT1
PSSTARTGRADIENT10
PSSTARTGRADIENT11
PSSTARTGRADIENT12
PSSTARTGRADIENT13
PSSTARTGRADIENT14
PSSTARTGRADIENT15
PSSTARTGRADIENT16
PSSTARTGRADIENT17
PSSTARTGRADIENT2
PSSTARTGRADIENT3
PSSTARTGRADIENT4
PSSTARTGRADIENT5
PSSTARTGRADIENT6
PSSTARTGRADIENT7
PSSTARTGRADIENT8
PSSTARTGRADIENT9
PSSTARTGRADIENTBACKGROUND
PSTASKDEPENDENCYLINE
PSVERTICALGRIDLINES

PSCHARTSTYLE style sheet

Chart Colors

The following table lists all the colors that are defined in the PSCHARTSTYLE style sheet.

You assign colors to your chart using the Chart class and Gantt class color methods. You can use either the numeric or constant value.

Charts that have RevertToPre850 set to True use solid colors. Beginning with 8.50, charts use gradients.

Numeric Value	***Constant Value***	***Description***
–1	%ChartColor_Series_Default	Default series color. When a data point is set to this value, the default series color or the color set by the user for that series is used instead. This value works only with the SetColorArray method.
0	%ChartColor_Black	Black
1	%ChartColor_Blue	Blue
2	%ChartColor_Cyan	Cyan
3	%ChartColor_DarkGray	DarkGray
4	%ChartColor_Gray	Gray
5	%ChartColor_Green	Green
6	%ChartColor_LightGray	LightGray
7	%ChartColor_Magenta	Magenta
8	%ChartColor_Orange	Orange
9	%ChartColor_Pink	Pink
10	%ChartColor_Red	Red
11	%ChartColor_White	White

Numeric Value	***Constant Value***	***Description***
12	%ChartColor_Yellow	Yellow
13	%ChartColor_Red_Orange	Red_Orange
14	%ChartColor_Yellow_Green	Yellow_Green
15	%ChartColor_Blue_Violet	Blue_Violet
16	%ChartColor_Purple	Purple
17	%ChartColor_Yellow_Orange	Yellow_Orange

See Chapter 11, "Charting Classes," SetColorArray, page 433 and Chapter 11, "Charting Classes," SetDataColor, page 438.

See Chapter 11, "Charting Classes," SetActualTaskBarColor, page 480 and Chapter 11, "Charting Classes," SetPlannedTaskBarColor, page 487.

Considerations for Gantt Charts

In Gantt charts, the PSCHARTDEFAULT style class uses an 8-point font with a normal weight. This style class affects all text in the chart area, such as task labels, DateTime X-axis labels, and hints. The PSCHARTGANTTGRID style class uses a 10-point font with a normal weight, and it affects all text in the table area. For the font size between the table and chart areas to be the same, the point size in the PSCHARTGANTTGRID style class should be 1.25 times the point size of the font defined in PSCHARTDEFAULT. Note, however, that the font size for PSCHARTGANTTGRID should not exceed 13; otherwise, column text may bleed into another column.

OrgChart Class and RatingBoxChart Class Style Sheets

OrgChart class and RatingBoxChart class chart appearance attributes are controlled entirely using style classes.

To modify a style class you can either create a new style sheet or set style properties to new style classes.

To create a new style sheet:

1. Clone the style sheet.

   For example, in Application Designer, open the PSORGCHART style sheet and save as MYORGCHART.

2. Double-click a style class name and modify the values in the Class properties dialog.

3. Assign the new style sheet to the page on which the chart is placed.

   In Application Designer, open the page and select File, Definition Properties.

   On the Use tab, in the Page Style Sheet field, select the new style sheet.

To set style properties to new style classes:

1. Copy and paste to create a new style class within the style sheet that is associated with the chart.

   For example, copy and paste the PT_ORGCHART_TITLE style class and name it MY_ORGCHART_TITLE.

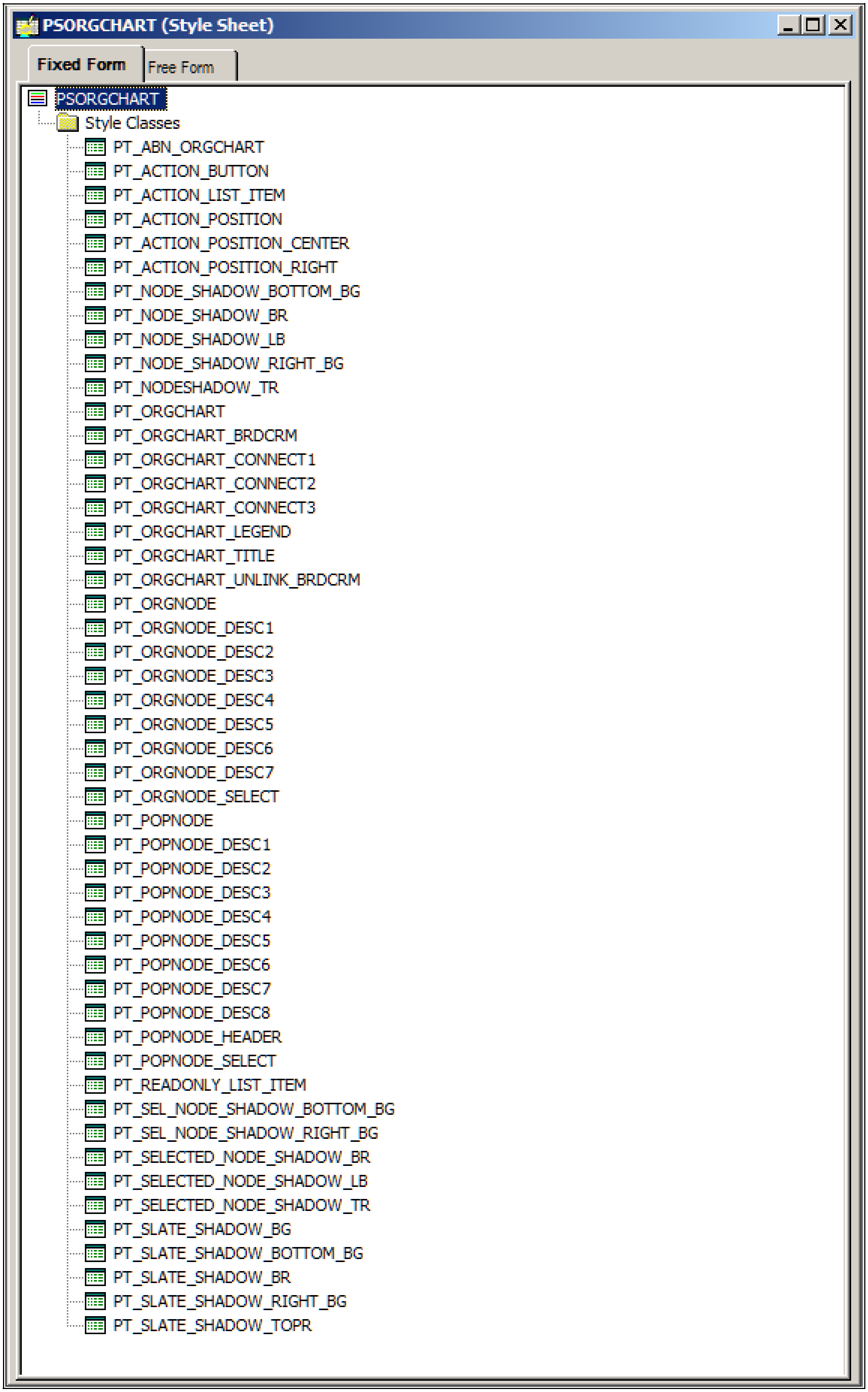
2. Modify the new style class.

3. Set the style property of the chart class to the new style class.

   For example:

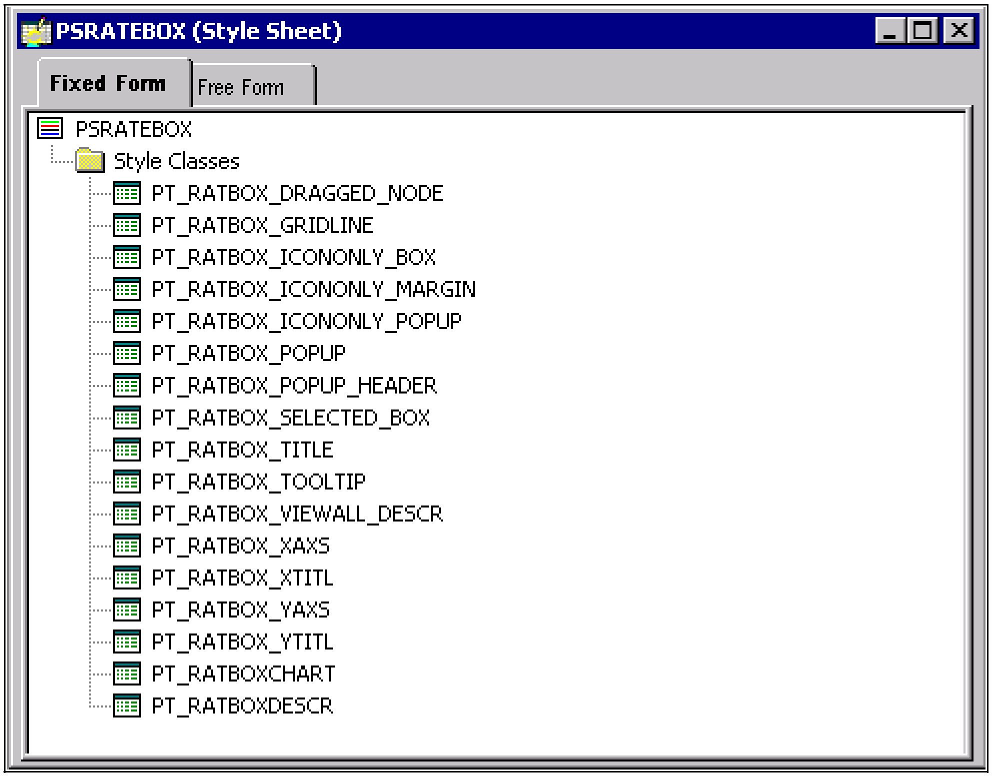
   ```
   &ocMyOrgChart.MainTitleStyle = MY_ORGCHART_TITLE;
   ```

The PSORGCHART style sheet controls the appearance of organization charts.


PSORGCHART (Style Sheet)
Fixed Form
Free Form
PSORGCHART
Style Classes
PT_ABN_ORGCHART
PT_ACTION_BUTTON
PT_ACTION_LIST_ITEM
PT_ACTION_POSITION
PT_ACTION_POSITION_CENTER
PT_ACTION_POSITION_RIGHT
PT_NODE_SHADOW_BOTTOM_BG
PT_NODE_SHADOW_BR
PT_NODE_SHADOW_LB
PT_NODE_SHADOW_RIGHT_BG
PT_NODESHADOW_TR
PT_ORGCHART
PT_ORGCHART_BRDCRM
PT_ORGCHART_CONNECT1
PT_ORGCHART_CONNECT2
PT_ORGCHART_CONNECT3
PT_ORGCHART_LEGEND
PT_ORGCHART_TITLE
PT_ORGCHART_UNLINK_BRDCRM
PT_ORGNODE
PT_ORGNODE_DESC1
PT_ORGNODE_DESC2
PT_ORGNODE_DESC3
PT_ORGNODE_DESC4
PT_ORGNODE_DESC5
PT_ORGNODE_DESC6
PT_ORGNODE_DESC7
PT_ORGNODE_SELECT
PT_POPNODE
PT_POPNODE_DESC1
PT_POPNODE_DESC2
PT_POPNODE_DESC3
PT_POPNODE_DESC4
PT_POPNODE_DESC5
PT_POPNODE_DESC6
PT_POPNODE_DESC7
PT_POPNODE_DESC8
PT_POPNODE_HEADER
PT_POPNODE_SELECT
PT_READONLY_LIST_ITEM
PT_SEL_NODE_SHADOW_BOTTOM_BG
PT_SEL_NODE_SHADOW_RIGHT_BG
PT_SELECTED_NODE_SHADOW_BR
PT_SELECTED_NODE_SHADOW_LB
PT_SELECTED_NODE_SHADOW_TR
PT_SLATE_SHADOW_BG
PT_SLATE_SHADOW_BOTTOM_BG
PT_SLATE_SHADOW_BR
PT_SLATE_SHADOW_RIGHT_BG
PT_SLATE_SHADOW_TOPR

PSORGCHART style sheet

The PSRATEBOX style sheet controls the appearance of rating box charts.



PSRATEBOX style sheet

Using the Chart Class

Use the Chart class to create charts that display data series in several common formats, including bar charts, line charts, pie charts, and bubble charts.

This section discusses:

- Chart terms.
- Creating charts using the Chart class.
- Chart class chart types.
- Data type of a Chart object.
- Scope of a Chart object.
- Error handling.
- Chart class methods and properties by category.

Chart Terms

The following is a list of Chart class terms and their descriptions:

Label	Text that identifies the data on one of the axes.
Legend	Text that identifies the different series in the chart.
Series	A grouping of related information. For example, if you were tracking sales for several divisions over many years, each division could be a series. Generally, every series in a chart has a distinct color. You can have more than one series in a chart for comparison. Each line in a line chart represents one series.
Overlay	Related data represented by a line drawn over the background chart. You can also specify data series, with different line types for each overlay.
Title	The three titles are Main, X axis, and Y axis. Each title identifies a portion of the chart.
TrueXY	A type of line chart that uses a numeric X axis instead of a categorical X axis. The numeric X axis supports non-uniform X data, that is, each series need not have the same number of points, and those points along the X axis need not match up across the series.
X axis	The axis that data is measured against.
Y axis	The axis that contains the data. In most charts, this is the vertical axis. In a horizontal bar chart, this is the horizontal axis.

Creating Charts Using the Chart Class

You can specify a rowset and have it graphed using a chart object with minimal PeopleCode. Within this rowset, one column must contain data for the X axis and another column must contain data for the Y axis. If more than one series of data is used, an additional column is required for each series. You can also set color and pop-up text for each point in this rowset using additional columns.

The following distinguishable types of data can be included in a chart:

- Primary data.
- Secondary or overlay data.

Overlay data may be quite different from primary data, so an additional rowset is needed.

The only object used with the Chart class is a chart object. Sub-objects do not exist. All methods and properties are used with the Chart object.

For a conceptual overview, a chart is easier to view in pieces; however, these pieces do *not* represent sub-objects.

A chart contains the following major parts:

- X axis
- Y axis
- Overlay
- Legend
- Data
- Title

The X, Y, and Overlay axes have access to titles and labels associated with their data.

Chart Class Chart Types

The following chart types are available for the Chart class:

- 2D Bar
- 3D Bar
- 2D Histogram
- 2D Percent Bar
- 3D Percent Bar
- 2D Stacked Bar
- 3D Stacked Bar
- 2D Horizontal Bar
- 2D Horizontal Percent Bar
- 2D Horizontal Stacked Bar
- 2D Line
- 2D Scatter
- 2D Pie
- 3D Pie

See Also

Chapter 11, "Charting Classes," Creating a Chart Using the Chart Class, page 557

Data Type of a Chart Object

Chart objects are declared using the Chart data type. For example,

```
Local Chart &MyChart;
```

```
Component Chart &Abs_Hist_Chart;
```

Scope of a Chart Object

A chart object can be instantiated only from PeopleCode.

You can use this object only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message notification, a Component Interface, and so on.

Error Handling

The PeopleCode program is terminated if a field or record is missing at runtime.

If a valid record is specified but no data is found, or for any other error, the chart is replaced by the message "Image creation failed."

If the Java Runtime Environment (JRE) is missing, an error appears and is logged.

Chart Class Methods and Properties by Category

The following topics subdivide the Chart class methods and properties by functional category.

The categories are divided among the different parts of a chart. However, all methods and properties are used with the chart object; a chart has no sub-objects.

Chart Data Methods and Properties

Use chart data methods and properties to set X and Y axes data.

See Chapter 11, "Charting Classes," Reset, page 433; Chapter 11, "Charting Classes," SetData, page 435; Chapter 11, "Charting Classes," SetDataHints, page 439; Chapter 11, "Charting Classes," SetDataSeries, page 443; Chapter 11, "Charting Classes," SetDataURLs, page 443; Chapter 11, "Charting Classes," SetDataXAxis, page 444; Chapter 11, "Charting Classes," SetDataYAxis, page 446; Chapter 11, "Charting Classes," DataStartRow, page 454; Chapter 11, "Charting Classes," DataWidth, page 454; Chapter 11, "Charting Classes," ImageMap, page 457 and Chapter 11, "Charting Classes," IsTrueXY, page 458.

Chart Appearance Properties

Use chart appearance properties to set the type of graph, the line style, and so on.

See Chapter 11, "Charting Classes," GridLines, page 455; Chapter 11, "Charting Classes," GridLineType, page 455; Chapter 11, "Charting Classes," Height, page 456; Chapter 11, "Charting Classes," IsDrillable, page 457; Chapter 11, "Charting Classes," IsPlainImage, page 458; Chapter 11, "Charting Classes," LineType, page 460; Chapter 11, "Charting Classes," RevertToPre850, page 463; Chapter 11, "Charting Classes," Style, page 464; Chapter 11, "Charting Classes," StyleSheet, page 464; Chapter 11, "Charting Classes," Type, page 465; Chapter 11, "Charting Classes," Width, page 466; Chapter 11, "Charting Classes," XRotationAngle, page 471; Chapter 11, "Charting Classes," YRotationAngle, page 476 and Chapter 11, "Charting Classes," ZRotationAngle, page 476.

Note. A new property, RevertToPre850, enables you to maintain the pre-8.50 visual style of an existing chart. If you want an existing chart to use pre-8.50 style classes, you must set the chart property RevertToPre850 to *True*.

See Chapter 11, "Charting Classes," Using Style Sheets with RevertToPre850, page 384.

Chart Color Methods

Use chart color methods to set and manipulate colors for either a series or for each point.

See Chapter 11, "Charting Classes," SetColorArray, page 433 and Chapter 11, "Charting Classes," SetDataColor, page 438.

Note. A new property, RevertToPre850, enables you to maintain the pre-8.50 visual style of an existing chart. If you want an existing chart to use pre-8.50 style classes, you must set the chart property RevertToPre850 to *True*.

See Chapter 11, "Charting Classes," Using Style Sheets with RevertToPre850, page 384.

Chart Axis Methods and Properties

Use chart axis methods and properties to further control the appearance of axes.

See Chapter 11, "Charting Classes," SetXAxisLabels, page 452; Chapter 11, "Charting Classes," SetYAxisLabels, page 452; Chapter 11, "Charting Classes," XAxisCross, page 466; Chapter 11, "Charting Classes," XAxisLabelOrient, page 468; Chapter 11, "Charting Classes," XAxisStyle, page 470; Chapter 11, "Charting Classes," XAxisTicks, page 470; Chapter 11, "Charting Classes," YAxisLabelOrient, page 472; Chapter 11, "Charting Classes," YAxisMax, page 473; Chapter 11, "Charting Classes," YAxisMin, page 473; Chapter 11, "Charting Classes," YAxisStyle, page 474 and Chapter 11, "Charting Classes," YAxisTicks, page 475.

Chart Title Properties

Use chart title properties to set the text and style of titles for the graph and the axes.

See Chapter 11, "Charting Classes," MainTitle, page 461; Chapter 11, "Charting Classes," MainTitleOrient, page 461; Chapter 11, "Charting Classes," MainTitleStyle, page 461; Chapter 11, "Charting Classes," XAxisTitle, page 470; Chapter 11, "Charting Classes," XAxisTitleOrient, page 471; Chapter 11, "Charting Classes," XAxisTitleStyle, page 471; Chapter 11, "Charting Classes," YAxisTitle, page 475; Chapter 11, "Charting Classes," YAxisTitleOrient, page 475 and Chapter 11, "Charting Classes," YAxisTitleStyle, page 476.

Chart Legend Methods and Properties

Use chart legend methods and properties for each series legend.

See Chapter 11, "Charting Classes," SetLegend, page 446; Chapter 11, "Charting Classes," HasLegend, page 456; Chapter 11, "Charting Classes," LegendMaxEntries, page 459; Chapter 11, "Charting Classes," LegendPosition, page 459 and Chapter 11, "Charting Classes," LegendStyle, page 460.

Chart Overlay Methods and Properties

Use chart overlay methods and properties to set and manipulate overlay data.

See Chapter 11, "Charting Classes," SetOLData, page 447; Chapter 11, "Charting Classes," SetOLDataSeries, page 450; Chapter 11, "Charting Classes," SetOLDataXAxis, page 450; Chapter 11, "Charting Classes," SetOLDataYAxis, page 451; Chapter 11, "Charting Classes," OLLineType, page 462 and Chapter 11, "Charting Classes," OLType, page 462.

Scatter Chart Methods and Properties

Use scatter chart methods and properties to set and manipulate scatter charts and bubble charts.

See Chapter 11, "Charting Classes," SetDataAnnotations, page 437; Chapter 11, "Charting Classes," SetDataGlyphScale, page 438; Chapter 11, "Charting Classes," SetOLDataAnnotations, page 448; Chapter 11, "Charting Classes," SetOLDataGlyphScale, page 449 and Chapter 11, "Charting Classes," ShowCrossHair, page 464.

TrueXY Line Chart Properties

Use trueXY line chart properties to set and manipulate trueXY line charts.

See Chapter 11, "Charting Classes," XAxisCrossPoint, page 467; Chapter 11, "Charting Classes," XAxisMax, page 468; Chapter 11, "Charting Classes," XAxisMin, page 468; Chapter 11, "Charting Classes," XAxisPrecision, page 469; Chapter 11, "Charting Classes," XAxisScaleResolution, page 469; Chapter 11, "Charting Classes," YAxisCrossPoint, page 472; Chapter 11, "Charting Classes," YAxisMax, page 473; Chapter 11, "Charting Classes," YAxisMin, page 473; Chapter 11, "Charting Classes," YAxisPrecision, page 473 and Chapter 11, "Charting Classes," YAxisScaleResolution, page 474.

Using the Gantt Class

Use the Gantt class to create Gantt charts. A Gantt chart displays tasks along a time line. Gantt charts are frequently used in project management because they provide a graphical illustration of a schedule, which helps in planning, coordinating, and tracking project tasks.

This section provides an overview of Gantt chart terminology and discusses how to:

- Use Gantt charts in PeopleSoft Pure Internet Architecture.
- Create Gantt charts.
- Specify DateTime axis formats.

- Work with start and end dates.
- Use Gantt glyphs.
- Use style sheets with Gantt charts.

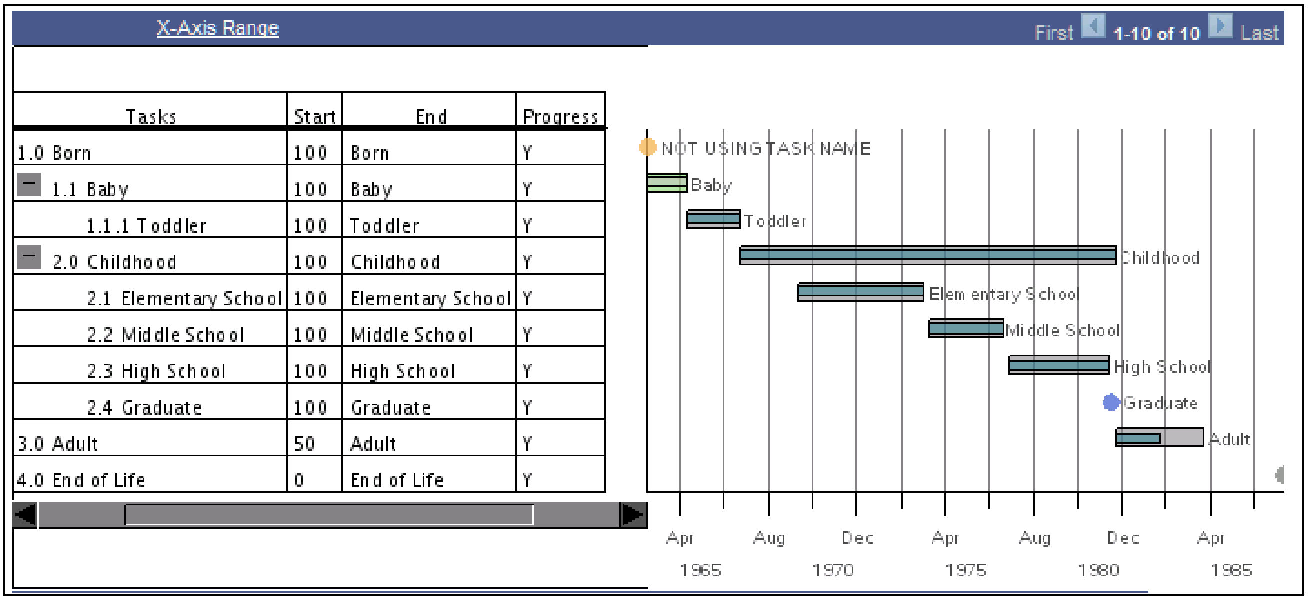
Understanding Gantt Chart Terms

Gantt charts can be composed of many different elements, such as a title, a label, and so on. The following terms apply specifically to Gantt charts:

Activity or**Task**	See Task.
Baseline	A point-in-time snapshot of the project plan that allows the user to identify how the project plan has changed since that point in time.
Data area	The part of the chart that graphically shows the span of dates that each task (or activity) covers. (This area is the right side of the displayed chart.)
Dependencies	A relationship between one task and other tasks that drives when tasks can begin and end. A dependent task cannot start before the task it is dependent on is either started or completed (depending on business rules defined in the application).
Milestone	A specialized mark on the chart that names a meaningful point in time.
Resource	A source of supply or support that is assigned to work on an activity or task.
Taskor **Activity**	In a Gantt chart, each row (task or activity), is represented in the Y axis of the chart, and the span of dates for each row is displayed on the X axis.
X axis	The X axis displays dates that are spanned by items in the Y axis.
X-axis labels	The part of the chart that lists the time scale. The X-axis labels support multiple levels of granularity: second, minute, hour, day, week, month, year.
Y axis	The Y axis is the display of tasks or activities, and is displayed as rows.
Y-axis labels	The part of the chart that lists activities and their attributes. Activities are shown hierarchically, and subtasks can be hidden or displayed by expanding or collapsing higher level tasks.

Using Gantt Charts in PeopleSoft Pure Internet Architecture

The Gantt chart consists of two sections: a table section and a chart section.



Example Gantt chart

The left side of the chart, where you see the table, is the table section. The other side, where you see the bars, is the chart section.

Horizontal scrollbars below the table section support scrolling through the task-related columns; below the task section, they support scrolling through the task bars.

The amount that a section scrolls depends on the value set for the SetTableXScrollBar method. Each click on the arrow scrollbar scrolls the task either left or right by the amount set with the SetTableXScrollBar method (or 10 percent of the section area if a value is not set).

See Chapter 11, "Charting Classes," SetTableXScrollbar, page 489.

Vertical scrollbars support scrolling through the task hierarchy.

Table Section

The table section contains all of the tasks and associated subtasks, and displays them in a hierarchy. Tasks that contain subtasks are called *parent tasks,* and subtasks are called *child tasks.* You can click on the expand (collapse) image to the left of the task name (Baby, above) to expand or collapse the subtask hierarchy. Note that subtasks may also act as parent tasks to other subtasks.

Each task has a name (Born, Baby, Childhood, and so on). It also has a level. Parent tasks have a higher level than child tasks.

Chart Section

The chart section displays the tasks, task dependencies, and milestones graphically.

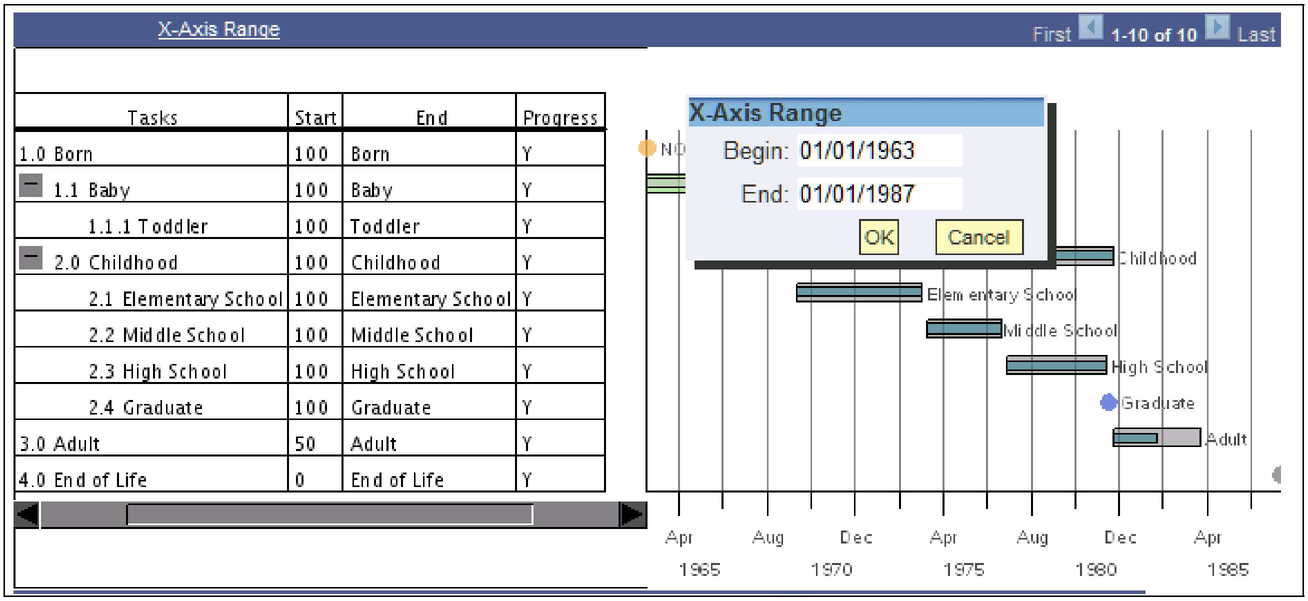
In the previous example, each horizontal bar represents a single task. The progress bar (that is, the percentage complete for a particular task) is indicated with a bar of a different color above the task bar. The milestone date (Graduate in the previous example) is represented with a circle.

You can also have dependencies between tasks. These are represented as lines connecting the task bars.

You can execute custom FieldChange PeopleCode when a user clicks a task bar or a task dependency line.

X-Axis Range

Click the X-Axis Range link on the Gantt toolbar to change the Begin and End dates for the X axis.



Example of X-Axis range dialog

User Interaction

A user can interact directly with a chart to change the underlying chart data.

These user actions are supported:

- Resize the task bar.
- Reposition the task bar.
- Reposition a milestone glyph.
- Move the separator between the table section and the chart section.
- Update task details, task start time and end time, and the progress bar.

When a user changes values in the chart section by dragging and dropping bars, the Planned Start, Planned End, and Progress values in the grid column will also be updated if the grid and the chart section use the same record and fields.

You can control the level the user is able to interact with the chart using the Gantt class properties InteractiveStart, InteractiveEnd, InteractiveProgress, and InteractiveMove.

See Chapter 11, "Charting Classes," InteractiveStart, page 511; Chapter 11, "Charting Classes," InteractiveEnd, page 510; Chapter 11, "Charting Classes," InteractiveProgress, page 511 and Chapter 11, "Charting Classes," InteractiveMove, page 511.

Creating Gantt Charts Using the Gantt Class

Every Gantt chart has at least one data set used to define the tasks and the information related to each task, such as start date, end date, milestones, percent finished, and so on.

A second data set can be used to describe dependencies between tasks.

The following methods are required for using the Gantt chart:

- SetTaskData

  Use this method to specify where most of the information for the Gantt chart is stored. You can specify either a rowset or a record.

- SetTaskID

  Use this method to specify the task ID, or name, of the task. Every task must have a unique task identifier. The task ID is used to support task linking and dependencies.

- SetPlannedStartDate

  Use this method to specify the planned starting date of the task. Each task must have its own planned starting date.

- SetPlannedEndDate

  Use this method to specify the planned ending date of the task. Each task must have its own planned ending date.

Though not required, Oracle recommends using the SetTaskName method to display meaningful information in the table section of the Gantt chart.

If you only use the required methods and do not use the SetTaskAppData method, Oracle recommends that you also use the SetChartArea method and dedicate most, if not all, of the entire area to the chart, and not the table.

If you specify one actual date, either for start or end, then you must specify the other (SetActualStartDate, SetActualEndDate).

In addition, if you want to use dependency data, the following methods are required:

- SetTaskDependencyData

  Use this method to specify where most of the information for the dependency data is stored. You can specify either a rowset or a record.

- SetTaskDependencyParentID

  In a dependency, one task depends on another. Use this method to specify the parent task, that is, the one that the other (child) task depends upon.

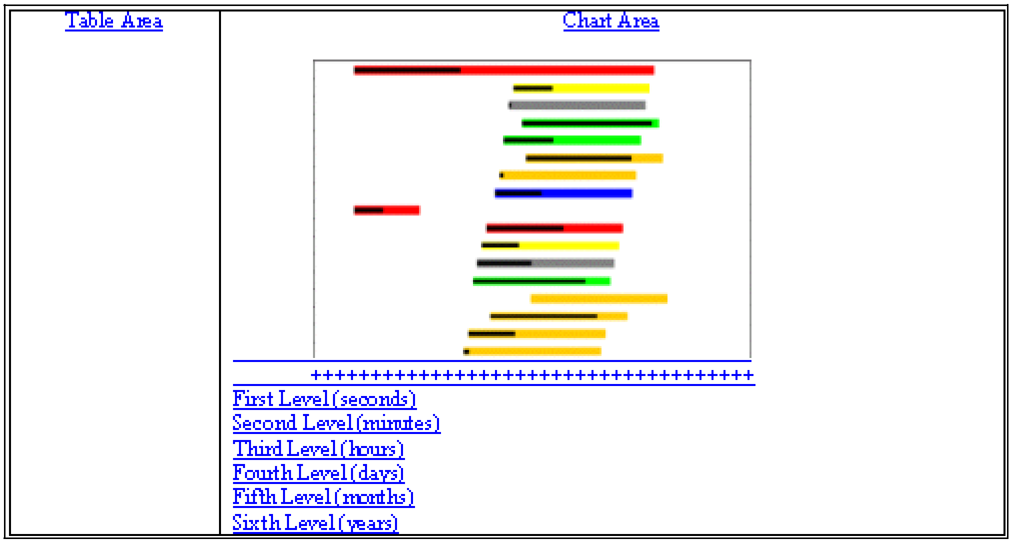
- SetTaskDependencyChildID

  In a dependency, one task depends on another. Use this method to specify the child task, that is, the one that depends on another.

Specifying DateTime Axis Formats

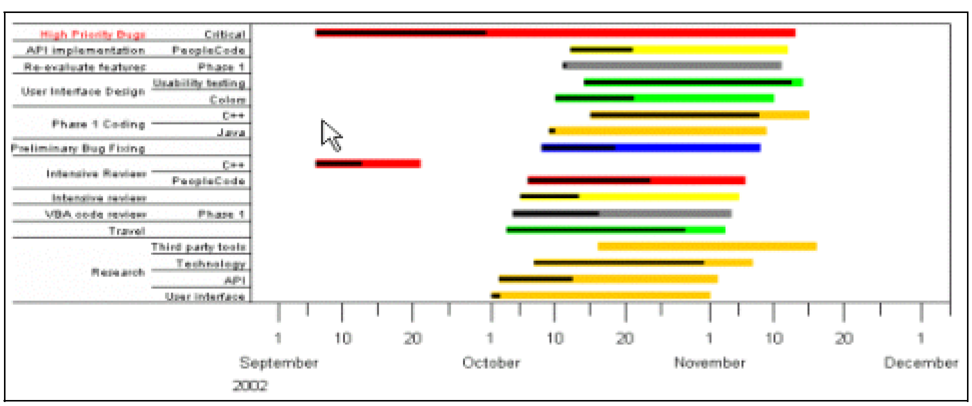
You can control the display of the label format that describes the DateTime axis in up to six different time increments, from seconds to years, or any combination thereof simultaneously for a given time period. Each time increment, if enabled through the DateTimeAxis specific methods, occupies a single level within the X-axis hierarchy. Smaller time increments display closer to the axis, while larger time increments display farther from the axis.

The following example shows the levels of the labels for the DateTime axis. The first level, in seconds, is the closest level to the chart edge; next are minutes, at the second level; and then hours, at the third level, and so on. Generally, the smallest unit should be on the lowest level. Therefore, if months and years appear, then months appear at the first level and years appear at the second level.



Example of levels of DateTime axis labels

In the following example, days, months, and years are defined at levels 1, 2, and 3, respectively. Therefore, the days of the month (1, 10, 20) appear closest to the chart area, the months appear below them, and the years appear below that.



DateTime format example

By default, the DateTime axis displays both the year and the months, with the major tick mark marking the months.

Note. The charting algorithm may or may not display a given time increment, even if it is enabled. The resolution of the given time increments may be too great, depending on the length of the time period defined in the chart section.

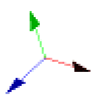
Working with Start and End Dates

You must set the end date to be after the start date, for both the planned as well as actual dates, or else you will receive an error.

Both the start and end dates you specify for a task must fall within the axis dates, that is, the dates specified by the AxisEndDateTime and AxisStartDateTime properties. If you specify task dates that fall outside of the axis dates, the task appears in the table section of the Gantt chart, but no bar appears in the chart section of the Gantt chart, even if one of the dates falls *inside* the axis dates.

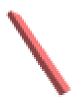
Using Gantt Glyphs

You can use all of the following glyphs with a Gantt chart. You can use either the numeric or constant value.



Constant: %ChartGlyph_Axe

Numeric value: 1

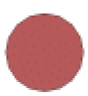


Constant: %ChartGlyph_Bar

Numeric value: 2

Constant: %ChartGlyph_Box

Numeric value: 3

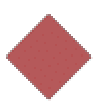


Constant: %ChartGlyph_Circle

Numeric value: 4

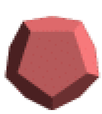
Constant: %ChartGlyph_Cross

Numeric value: 5



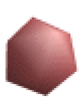
Constant: %ChartGlyph_Diamond

Numeric value: 6



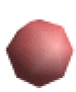
Constant: %ChartGlyph_Dodecahedron

Numeric value: 7



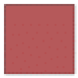
Constant: %ChartGlyph_Icosahedron

Numeric value: 8



Constant: %ChartGlyph_Sphere

Numeric value: 9

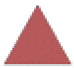


Constant: %ChartGlyph_Square

Numeric value: 10

Constant: %ChartGlyph_Star

Numeric value: 11



Constant: %ChartGlyph_Triangle

Numeric value: 12

Using Style Sheets with Gantt Charts

In Gantt charts, the PSCHARTDEFAULT style class uses an 8-point font with a normal weight. This style class affects all text in the chart area, such as task labels, DateTime X-axis labels, and hints. The PSCHARTGANTTGRID style class uses a 10-point font, also with a normal weight, and affects all text in the table area. For the font size between the table and chart areas to be the same, the point size in the PSCHARTGANTTGRID style class should be 1.25 times the point size of the font defined in PSCHARTDEFAULT. Note, however, that the font size for PSCHARTGANTTGRID should not exceed 13; otherwise, column text may bleed into another column.

Data Type of a Gantt Object

Gantt objects are declared using the Gantt data type. For example,

```
Local Gantt &MyGantt;
```

Scope of a Gantt Object

A Gantt object can be instantiated only from PeopleCode.

You can use this object only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message notification, a Component Interface, and so on.

Error Handling

The PeopleCode program is terminated if a field or record is missing at runtime.

If a valid record is specified but no data is found, or for any other error, the chart is replaced by the message "Image creation failed."

If the Java Runtime Environment (JRE) is missing, an error appears and is logged.

Using the OrgChart Class

Use the OrgChart class to create organization charts. An organization chart represents hierarchical data as boxes arrayed in levels, which can be oriented top to bottom, with horizontal rows, or left to right, with vertical rows. Connectors show parent and child relationships between boxes on adjacent levels.

You can use this chart type to display many different types of hierarchical relationships, including, but not limited to:

- People in organizations
- Positions in organizations
- Departments in organizations
- Components in bills of materials

This section provides an overview of organization chart terminology and discusses how to:

- Use organization charts in PeopleSoft Pure Internet Architecture.
- Create organization charts.
- Organization chart actions and events.
- Organization chart subrecord definitions.
- Minimum methods.

Understanding Organization Chart Terms

Organization charts can be composed of many different elements, such as the main chart, nodes, connectors, pop-up nodes, and so on. These terms apply specifically to organization charts:

Breadcrumbs	You can configure an organization chart to optionally display links at the top of an organization chart. Typically, these are the names of people in an organization hierarchy.
Button	See Drop-down list box.
Child node	A node that is one level below its parent node (except for the top-level node). Every node except the top-level node has one parent node.
Connector	Lines that link the boxes in an organization chart. Connectors denote parent and child relationships. A connector also shows the relationship between a pop-up node and the predecessor node of the pop-up.

Descriptor	A text string that conveys information about a node. A main chart node has up to seven descriptors, and a pop-up node has up to eight descriptors. Each descriptor can have an image associated with it and can be configured as a link. On the main chart, a descriptor can also be configured as a button that is associated with a drop-down list box. FieldChange PeopleCode associated with a descriptor can invoke a pop-up, change the display characteristics of the chart, and perform other processing logic.
Drop-down list box	You can configure a descriptor as a button associated with a drop-down list box. A drop-down list box can be linkable or read-only. If the drop-down list box is linkable, PeopleCode related to the descriptor executes when a user clicks a list item.
Header	Information that appears at the top of a pop-up.
Level	An organization chart is made up of boxes, representing nodes, at different levels. The position of a node in a chart is determined by its parent node (PTPARENT_CHART_ND) and display order (PTND_DISPLAY_ORDER).
Main chart	The primary chart in an organization chart showing individual entities, such as employees, in boxes related to one another hierarchically. A user can invoke PeopleCode that displays a pop-up node by clicking a link in a box, or node, of the main chart or by clicking a link in a pop-up node.
Node	A representation of a single entity, an individual instance of data, in an organization chart. A node is represented as a box with vertical lines connecting it to its parent node and to its child nodes, if any.
Node record	A derived/work record that contains the properties and data values for each node on the main chart. The node record is created using a clone of the PTORGNODE_SBR subrecord.
Parent node	A node that has one or more subordinate nodes, or child nodes, one level below it. Each node can be a parent for other nodes.
Pop-up	A new window that displays information specifically related to a node on the main chart or another pop-up. A pop-up can be invoked from a link on a main chart node or from a link on a pop-up node. FieldChange PeopleCode controls the appearance and content of a pop-up.
Pop-up node	A single instance of data on a pop-up. A pop-up can have one or more nodes, all related to one node of the main chart or one node of another pop-up.

Pop-up node record

A derived/work record that contains the properties and data values for each pop-up node on the main chart.

The node record is created using a clone of the PTORGPOPUPNODE_SBR subrecord.

Using Organization Charts in PeopleSoft Pure Internet Architecture

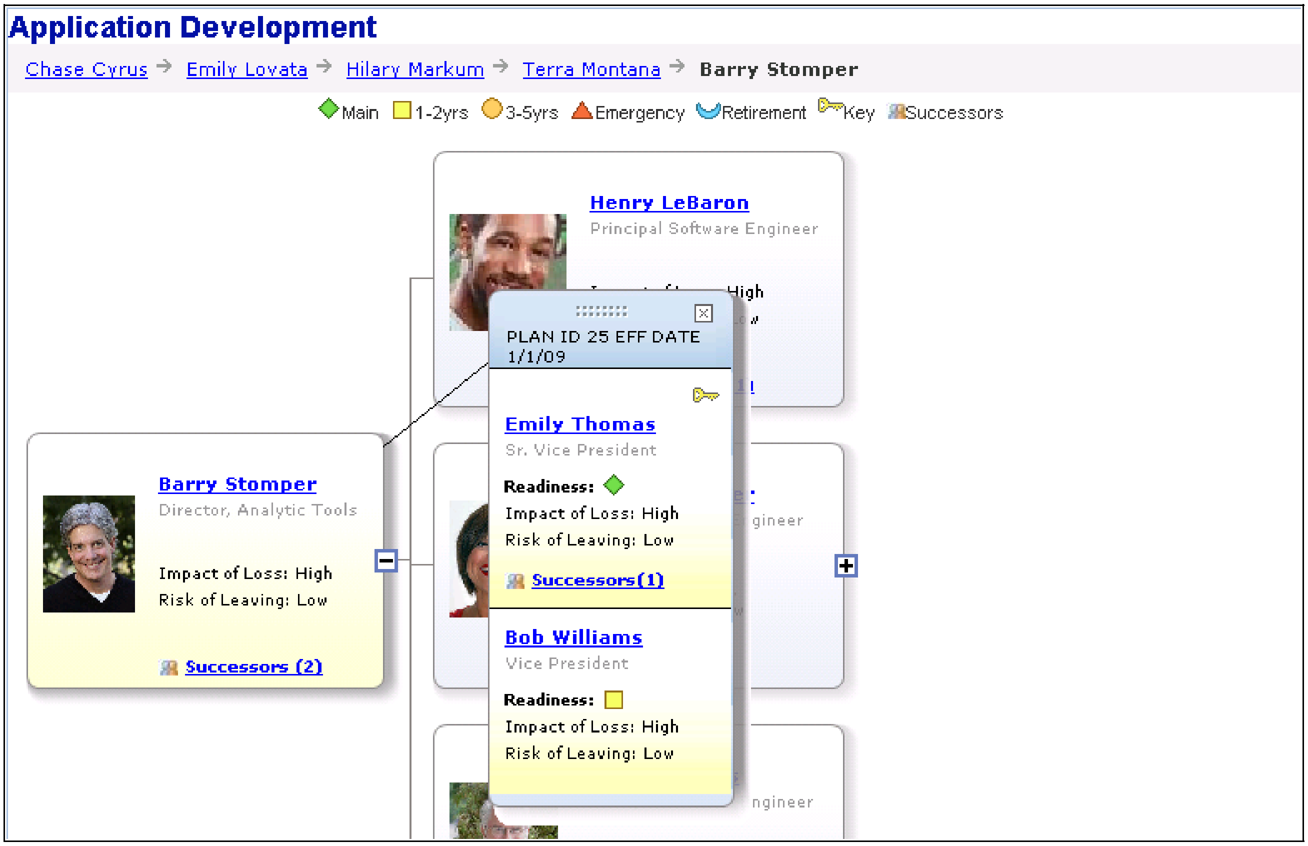
A PeopleSoft organization chart consists of two parts: the main hierarchical chart and pop-ups that enable the user to access additional related data for each node.

An organization chart can be oriented vertically or horizontally.

The following example shows a vertical organization chart:

Vertically oriented organization chart

The following example shows a horizontally oriented organization chart with a pop-up:



Horizontally oriented organization chart with a pop-up chart

The underlying chart is the main organization chart. The chart overlaying the nodes on the right is a pop-up chart.

Main Chart

The main chart can have multiple levels. Each level consists of one or more nodes. Each node represents an individual entity, such as an employee, in the organization. A node has one parent node (except the top-level node, which does not have a parent node) and zero, one, or more child nodes. Each node in the main chart has up to seven descriptors and each descriptor can be configured as a link or as a button associated with a drop-down list box. Application developers can associate FieldChange PeopleCode with each descriptor link or button associated with a linkable drop-down list box.

Internet Explorer 8 Considerations

In Microsoft Internet Explorer 8, organization charts may not display correctly when the browser Document mode is set to Standards mode. Only the Quirks mode is supported for PeopleSoft charts.

See See the TroubleShooting Browser Limitations document on Oracle's MyOracleSupport website.

Pop-up

A pop-up displays a pop-up node with information specifically related to the node from which it was invoked. A connector line visually links the pop-up chart to the predecessor node, the node in the chart from which the pop-up was invoked.

If the pop-up contains more nodes than can be displayed (that is, if MaxPopupDisplayNode is set to a number that is less than the total number of nodes) then a scroll bar appears in the pop-up. In some circumstances when a pop-up has a scroll bar and has more than one level, when a user clicks a link on a level other than the top level the new pop-up may not appear in the display area. If this occurs, the user can scroll down using the scroll bar in the initial pop-up to view the new pop-up node.

Note. The amount of time it takes for a pop-up to appear is proportional to the number of nodes in the pop-up. A pop-up with hundreds of nodes may take a long time to appear. An hourglass appears to let the use know the system is working to retrieve the nodes.

A pop-up is invoked through PeopleCode associated with the FieldChange event for a descriptor link on a main chart node or another pop-up node.

A pop-up consists of a header and one or more nodes with each node having up to eight descriptors. Each descriptor can be configured as a link.

The user can drag-and-drop a pop-up node to reposition it but the new position will not be retained after any action that requires a server trip that reloads the chart.

Creating Organization Charts Using the OrgChart Class

You can populate the organization node information using rowsets linked to existing record data, or you can populate the rowset from PeopleSoft trees using the Tree class APIs.

When you add an organization chart to a page in PeopleSoft Application Designer, you use the chart properties to associate the chart control with a chart record field. You will use this field name to identify the chart in PeopleCode.

You will need to create two records that will be the organization chart node records. The org node record holds the values for the properties and data for each of the nodes of the main chart. The pop-up node record holds the values for the properties and data for each of the pop-up nodes.

The organization chart node records must be in the component buffer at runtime. This means that you will have to place them on a page at level one in the component. You can hide them.

The maximum number of nodes you can create is limited according to browser, as shown in the following table:

Browser	***Maximum Nodes***
Microsoft Internet Explorer	250
Mozilla Firefox	1000
Apple Safari	600

The records can have any name, but they must include clones of the PeopleTools-delivered subrecords PTORGNODE_SBR and PTORGPOPUPNODE_SBR. Since the subrecords will carry the FieldChange PeopleCode for the chart, in most cases you should create unique subrecords for each chart.

Your PeopleCode will create and populate two rowsets . These must be component rowsets, linked to the node records of the chart, not standalone rowsets. The org node rowset has one row for each node in the organization chart, and the pop-up node rowset has one row for each pop-up node.

You place FieldChange PeopleCode on the fields of the node records to invoke pop-up charts, update displayed data, process application logic, and refresh the chart.

Using Images with Organization Charts

If your organization chart uses images, such as photographs, then the scroll area or grid with the fields that contain the images must have unlimited occurs count enabled. That is, the Unlimited Occurs Count check box must be selected on any scroll area or grid that includes a record with the fields from PTORGNODE_SBR or PTORGPOPUPNODE_SBR, but only when images are used.

Using Breadcrumbs with Organization Charts

You can configure an organization chart to display breadcrumbs. Typically, breadcrumbs display the names of people in an organization hierarchy. The following example shows an organization chart with breadcrumbs:

Organization chart with breadcrumbs

An organization chart can have only one set of breadcrumbs. The breadcrumbs appear horizontally at the top of an organization chart, above the chart legend if the legend is set. The breadcrumbs string wraps automatically if it is wider than the organization chart. If a breadcrumb contains a space, the breadcrumb string may wrap at the space, causing the breadcrumb to appear on two lines.

Style classes control the formatting of breadcrumbs. The default style classes are PT_ORGCHART_BRDCRM and PT_ORGCHART_UNLINK_BRDCRM. You can specify your own style classes using the CrumbDescrStyle and UnlinkCrumbDescrStyle properties.

You can specify an image to separate breadcrumb entries. If an image is not specified, the breadcrumbs are separated by three spaces. Each breadcrumb entry can be configured to be linkable. This configuration is similar to linkable text on organization chart nodes.

To implement breadcrumbs in an organization chart, you:

- Clone the breadcrumb subrecord PTORGCRMB_SBR.
- Add the clone of PTORGCRMB_SBR to a work record.
- Add the work record to a page such that it is available in the component buffer so that PeopleCode can be invoked from the record fields.

  One way to accomplish this is to include the record in a level 1 grid on the same page as the organization chart, and hide the grid.
- To make a breadcrumb linkable, set the corresponding PTORGCRMBLINKABLE field to "Y", in much the same way as you make a descriptor linkable in an organization chart node.
- Add PeopleCode to provide application-specific processing when a linkable breadcrumb is selected.

  When a user clicks a linkable breadcrumb, any FieldEdit and FieldChange PeopleCode in the PTORGCRMBID field executes.

At runtime, your PeopleCode will add rows to the work record that holds the information for each breadcrumb (person).

Each breadcrumb in a chart is defined by a row in the record.

See Chapter 11, "Charting Classes," Organization Chart Subrecord Definitions, page 417 and Chapter 11, "Charting Classes," CrumbDescrStyle, page 528.

Breadcrumb Methods and Properties

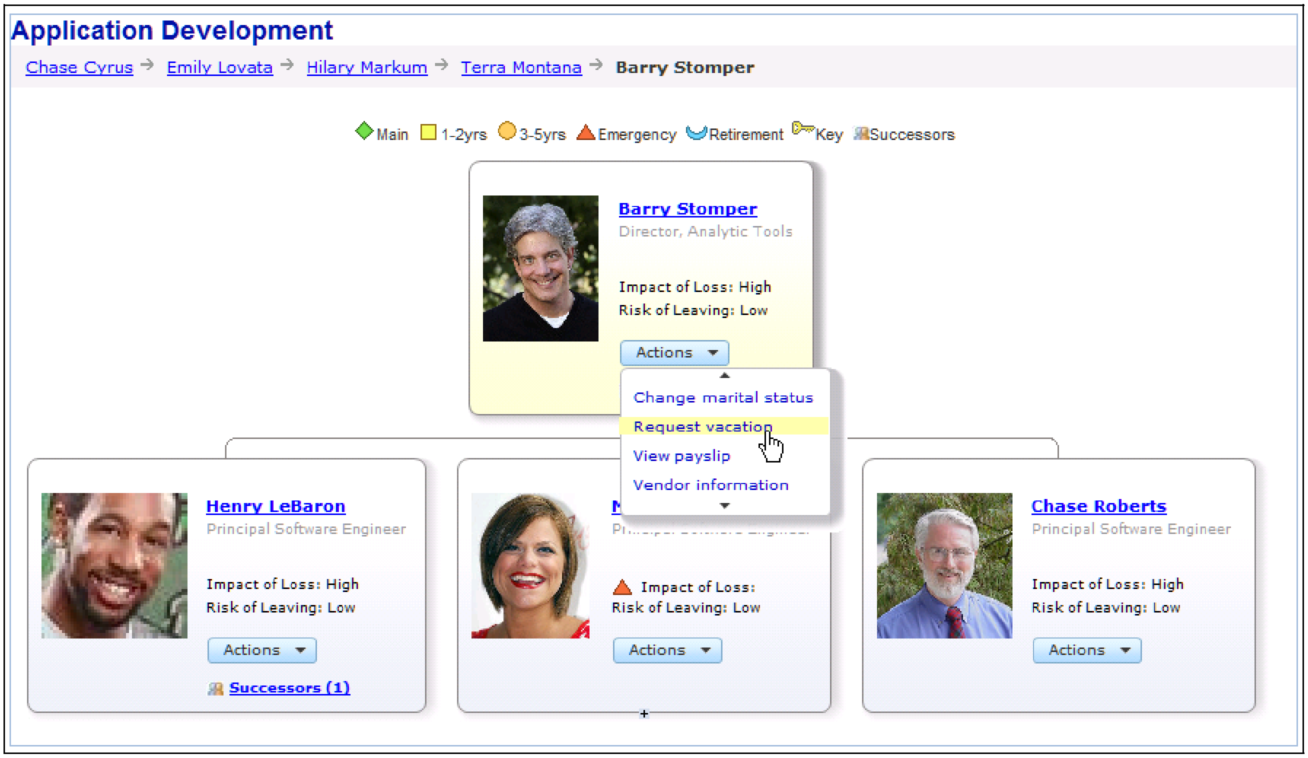
The following OrgChart class methods and properties control the appearance of breadcrumbs:

- SetCrumbData
- SetCrumbRecord
- UnlinkCrumbDescrStyle
- CrumbDescrStyle
- CrumbMaxDisplayLength
- CrumbSeparatorImage

Using Drop-Down List Boxes

You can configure an organization chart to include buttons associated with drop-down list boxes for any of the node descriptors, enabling you to present related tasks for a node.

A drop-down list box can be linkable or read-only. If the drop-down list box is linkable, field change PeopleCode associated with the descriptor field, PTNODE_DESCR*n*, executes when a user clicks on a list item. If the drop-down list box is read-only, field change PeopleCode does not execute.



Example of an organization chart with buttons

The width of a drop-down list box is the width of the widest list item that the box contains.

The height of the list item box is defined dynamically. An up-and-down arrow vertical scroll appears when the number of list items is greater than the value specified for the MaxDropdownDisplayItem property. The default value is 7.

A scroll appears if the list extends beyond the visible chart area.

To implement drop-down list boxes:

- Include the subrecord PTORGBOXFLD_SBR at the end of the chart node record.
- Set the PTNDDESC*n*LINKABLE field to "D" for the descriptor that is to have the linkable drop-down list box. Set the field to "R" is the field is to be read-only.
- Populate the PTDESCR*n*_DROPDOWN field value in the updated rowset to provide the dropdown box ID for the descriptor that has the PTNDDESC*n*LINKABLE field set to "D" or "R".
- Create a derived/work record that contains the subrecord PTORGBOXLIST_SBR, and populate the corresponding rowset with the list information for each of the dropdown box IDs.
- Place FieldChange PeopleCode on the PTNODE_DESCR*n* field to perform corresponding logic when a list item is selected in the linkable drop-down list box.

Use the DropDownBoxStyle property to specify the style to specify the button position in the node.

Specifying Scroll Type

If an organization chart is larger than the viewable area, the user needs to scroll to view other areas of the chart. You can specify whether the organization chart will use scroll bars or alternative scrolling.

Example of an organization chart using alternative scrolling

If you specify alternative scrolling, a scroll navigator and a hand cursor appear instead of vertical and horizontal scroll bars.

The arrows on the scroll navigator enable the user to quickly position the right, left, top, or bottom of the chart in the viewable area.

If the user scrolls the chart, the center button of the scroll navigator returns the chart to its initial display position. If the user changes focus by selecting a different node, the chart centers on that node. The center button then returns the chart to display centered on the focus node.

Note. If the breadcrumbs and chart legend are so large that there is not sufficient room to display the scroll navigator in the upper left portion of the chart, it will appear in the lower left corner.

The hand cursor enables the user to grab the background of the chart and move it within the viewable area.

Note. When using the hand cursor to scroll, you must position the hand cursor in the chart area outside the chart nodes and popup nodes. Scrolling does not work if the hand cursor is positioned within a chart node or pop-up node.

Use the ChartScrollType property to specify which type of scrolling to use. The default is scroll bars.

Organization Chart Actions and Events

An organization chart can execute FieldEdit or FieldChange PeopleCode in response to these user actions:

- Select an expand/collapse icon.
- Select a descriptor link or linkable drop-down list item.
- Close a pop-up node.
- Select a breadcrumb.
- Click on a node image.
- Click on a node.

Selecting Expand/Collapse Icons

When the user clicks an expand/collapse icon, the system toggles the EXPANDED_FLAG field value from "Y" to "N" or from "N" to "Y" and changes the icon on the node to collapsed or expanded, correspondingly. This action triggers field change processing (FieldEdit and FieldChange events) on EXPANDED_FLAG. Typically, FieldChange PeopleCode checks the EXPANDED_FLAG and PARENT_FLAG of the node and performs these actions:

- If `EXPANDED_FLAG="Y"` and `PARENT_FLAG="X"`, loads the child rows into the rowset and sets `PARENT_FLAG="Y"`. The updates appear when the chart is refreshed.
- If `EXPAND_FLAG="N"` (collapse action), it does nothing.
- If `EXPAND_FLAG="Y"` (expand action) and `PARENT_FLAG="Y"` (child data is already in rowset), it does nothing.

Note. If your PeopleCode doesn't populate the CollapsedImage and ExpandedImage properties of the organization chart, then no expand/collapse icon appears on any nodes in the chart, and the expand/collapse action is disabled for the chart.

Selecting a Descriptor Link or Linkable Drop-Down List Item

When the user clicks a descriptor link or linkable drop-down list item, field change processing is triggered on the PTNODE_DESCR*n* field in the node record.

For instance, suppose descriptor 2 is linked to a pop-up chart. When the user clicks the descriptor 2 link, FieldChange PeopleCode on the PTNODE_DESCR2 field in the node record performs the following actions:

1. Sets the node's`PTPOPUPEXPAND_FLAG ="Y"`.
2. Sets the node's `PTPOPUPHDRMAINDESC`.
3. Sets the node's `PTPOPUPHDRMAINICON`.
4. If the node's `PTHASPOPUP_FLAG ="X"`, then populates the pop-up node rowset and sets `PTHASPOPUP_FLAG ="Y"` to indicate that the node's descriptor was clicked for the first time.
5. Refresh the chart with the pop-up displayed.

Closing a Pop-up Node

When the user closes a pop-up node it triggers field change processing on the PTPOPUPEXPAND_FLAG field.

FieldChange PeopleCode on the PTPOPUPEXPAND_FLAG field would need to restore any changes that were made to the originating node's display characteristics and reset the node's `PTPOPUPEXPAND_FLAG ="N"`.

Selecting a Breadcrumb

When a user clicks on a linkable breadcrumb, any FieldEdit and FieldChange PeopleCode in the PTORGCRMBID field executes. Typically PeopleCode is used to redraw the organization giving focus to the node that was clicked.

Clicking on a Photograph

When a user clicks on a node image, field change PeopleCode on the PTNDMAINICON_IMAGE field executes.

Clicking on a Node

When a user clicks on the body of a node -- not on a link, linkable drop-down list, or image -- field change PeopleCode on the PTCHART_NODE field executes.

Organization Chart Subrecord Definitions

This table describes the fields in PTORGNODE_SBR:

Field	***Description***
PTCHART_NODE	Specifies the node name. PTCHART_NODE and PTPARENT_CHART_ND are key fields of the PTORGNODE record.
PTPARENT_CHART_ND	Specifies the parent of the current chart node. If a node does not have a parent node defined, then the node is placed on the first level.
PTND_DISPLAY_ORDER	Specifies the display order of the sibling nodes for the same parent node. Nodes display left to right according to display order. By default, the nodes are ordered as they appear in the rowset.
PTCONNECTLINESTYLE	Specifies the style class name for connection line style. The default style class is PT_ORGCHART_CONNECT1.
PTNDMAINICON_IMAGE	Specifies the main icon image name for the node.

Field	***Description***
PTNODESTYLENAME	Indicates the style class name for the node to control node background color, border, style, and so on. The default style class is PT_ORGNODE.
PTFOCUS_FLAG	Indicates a focused node (that is, the node that has focus). Y – Focused node. N – Not a focused node. If more than one node is set to focused, only the first focused node in the rowset will be recognized as the focused node. The default is *"N"*.
PARENT_FLAG	Indicates whether a node is a parent. Y – The node is a parent and its direct children are already loaded into the rowset. X – The node is a parent and its direct children are not loaded in the rowset (that is, they will be loaded on demand). N – The node is not a parent. The default value is *"N"*.
EXPANDED_FLAG	Indicates whether the node is expanded or collapsed. Y – The chart shows the node expanded with the expanded image icon and its immediate children displayed. N – The chart shows the node collapsed with the collapsed image icon.
DISPLAYED_FLAG	Indicates whether the node displays in the chart. Y – The node displays if the parent node's `EXPANDED_FLAG ="Y"` (the parent node is not collapsed). N – This node and its child nodes do not display in the chart's display area. The default is *"Y"*.

Field	***Description***
PTHASPOPUP_FLAG	Indicates whether the node has a pop-up. Y – This node has a link that invokes a pop-up and the data for the pop-up is already loaded into the pop-up node rowset. X – This node has a link that invokes a pop-up and the data for the pop-up is not loaded in the rowset. The data will be loaded on demand. N – This node does not have a link to invoke a pop-up chart. The default value is *"N"*.
PTPOPUPEXPAND_FLAG	Indicates whether a node's pop-up nodes display. Y – Pop-up nodes display. N – No pop-up nodes display. If the user clicks the close icon in the pop-up node, the FieldChange event for PTPOPUPEXPAND_FLAG executes. The application sets the value of PTPOPUPEXPAND_FLAG to "N" when the pop-up node is closed.
PTNODE_DESCR*n*	Specifies the descriptor *n* of the node, where *n* is an integer from 1 to 7; for example, PTNODE_DESCR2. Maximum length is 50 characters.
PTNDDESC*n*LINKABLE	Specifies whether the descriptor *n* is linkable, a button with a linkable drop-down list box, or a button with a read-only drop down list box, where n is an integer from 1 to 8; for example, PTNDDESC2LINKABLE. Y – Linkable N – Not linkable D – Linkable drop down list box R – Read-only down list box The default value is "N". **Note.** If PTDESCR*n*_ICON_IMG is set to a valid image and PTNDDESC*n*LINKABLE is set to "D" or "R", drop down list box, the image is not displayed.
PTDESCR*n*_ICON_IMG	Specifies the name of the image associated with descriptor *n*, where *n* is an integer from 1 to 7; for example, PTDESCR2_ICON_IMG. **Note.** If PTDESCR*n*_ICON_IMG is set to a valid image and PTNDDESC*n*LINKABLE is set to "D" or "R", drop down list box, the image is not displayed.

Field	***Description***
PTDESCR*n*_ICON_POS	Indicates the position of the icon relative to descriptor *n*, where *n* is an integer from 1 to 7; for example, PTDESCR2_ICON_POS. L **–** To the left of the description. R **–** To the right of the description. The default value is based on user language. For instance, *"L"* for English and *"R"* for Hebrew.
PTADJUSTFORNULLS	Indicate whether to collapse empty attributes to conserve vertical space within the nodes. Y **–** Collapse. N **–** Do not collapse. The default value is *"N"*.
PTPOPUPHDRMAINDESC	Specifies the main descriptor of the pop-up chart header for this node
PTPOPUPHDRMAINICON	Specifies the name of the image to place on the top left of the pop-up chart header.
PTORGCHRTIMG	Specifies the node image. The maximum image size depends on your database platform. **Note.** If this field is placed on a grid and the column is hidden (Visible = False), the node images do not appear on the chart.
PSIMAGEVER	A unique number that is assigned to each image. This is a system-maintained value.
PTDESCR_UNDER_IMG	Determines whether descriptors appear to the side of the image or below it. Specify an integer from 1 to 7. The corresponding descriptor and all following descriptors appear below the image. All descriptors prior to the one specified appear to the side of the image. If no value is specified, or any value other than 1-7, the descriptors all appear to the side. If no image is displayed, this field is ignored.

This table describes the fields in PTORGPOPUP_SBR.

Field	***Description***
PTCHART_NODE	Specifies the name of the node that invoked this pop-up chart. PTCHART_NODE and PT_POP_UP_ID are keys for PTORGPOPUP_SBR.
PT_POP_UP_ID	The unique id for the pop-up . PTCHART_NODE and PT_POP_UP_ID are keys for PTORGPOPUP_SBR.
PTPREDECESSOR_NODE	Specifies the name of the predecessor node to the current chart node. The predecessor node is the node that invoked this node. Your application PeopleCode must set this value when the link is clicked on the predecessor node.
PTND_DISPLAY_ORDER	Sets the display order of the nodes within the same pop-up. By default the nodes are ordered as they appear in the rowset.
PTNDMAINICON_IMAGE	Specifies the main icon image name for the node.
PTNODESTYLENAME	Specifies the style class name for the node to control node background color, border, style, and so on. If no style class name is specified the PeopleTools default style class is used.
PT_POPUP_PARENT_ID	Set to the ID of the parent pop-up chart to which the PT_POP_UP_ID is attached. If the chart ID PT_POP_UP_ID is connected from the organization node, then its corresponding PT_POPUP_PARENT_ID value is NULL.
PTHASPOPUP_FLAG	Indicates whether the node has a pop-up. Y – This node has a link that invokes a pop-up and the data for the pop-up is already loaded into the pop-up node rowset. X – This node has a link that invokes a pop-up and the data for the pop-up is not loaded in the rowset. The data will be loaded on demand. N – This node does not have a link to invoke a pop-up chart. The default value is *"N"*.

Field	***Description***
PTPOPUPEXPAND_FLAG	Indicates whether a node's pop-up nodes display. Y – Pop-up nodes display. N – No pop-up nodes display. If the user clicks the close icon in the pop-up node, the FieldChange event for PTPOPUPEXPAND_FLAG executes. The application sets the value of PTPOPUPEXPAND_FLAG to *"N"* when the pop-up node is closed.
DISPLAYED_FLAG	Indicates whether the node will appear in the pop-up chart. Y = The node appears in the pop-up chart. N = The node never appears in the pop-up chart, including its pop-up nodes. Default is *"Y"*.
PTNODE_DESCR*n*	Specifies the descriptor *n* of the node, where *n* is an integer from 1 to 8; for example, PTNODE_DESCR2. Maximum length is 50 characters.
PTNDDESC*n*LINKABLE	Specifies whether the descriptor *n* is linkable, where *n* is an integer from 1 to 8; for example, PTNDDESC2LINKABLE. Y – Linkable N – Not linkable
PTDESCR*n*_ICON_IMG	Specifies the name of the image associated with descriptor *n*, where *n* is an integer from 1 to 8; for example, PTDESCR2_ICON_IMG.
PTDESCR*n*_ICON_POS	Indicates the position of the icon relative to descriptor *n*, where *n* is an integer from 1 to 8; for example, PTDESCR2_ICON_POS. L – To the left of the description. R – To the right of the description. The default value is based on user language. For instance, *"L"* for English and *"R"* for Hebrew.
PTADJUSTFORNULLS	Indicates whether to collapse empty attributes to conserve vertical space within the nodes. Y – Collapse N – Do not collapse

Field	***Description***
PTPOPUPHDRMAINDESC	Specifies the main descriptor of the pop-up chart header for this node. To force a line break in the header add a * * html tag. Example: `'PLAN ID 23  EFF DATE 1/1/09'` **Note.** *</br>* does not produce a line break. The html tag must be * *.
PTPOPUPHDRMAINICON	Specifies the name of the image to place on the top left of the pop-up chart header.

This table describes the fields in PTORGBOXFLD_SBR:

Field	***Description***
PTDESCR*n*_DROPDOWN	The drop down list ID for descriptor *n*, where *n* is an integer from 1 to 7; for example, PTDESCR2_DROPDOWN.
PTDES*n*_SLCT_ITM_ID	The selected list Item ID in the drop-down box for descriptor *n*, where *n* is an integer from 1 to 7; for example, PTDES2_SLCT_ITM_ID. The system sets this field to the value of the list item ID when the user clicks on a list item in the drop-down list box for descriptor *n*. You can add PeopleCode to the PTNODE_DESCR*n* field that performs business logic using the value in this field.

This table describes the fields in PTORGBOXLIST_SBR:

Field	***Description***
PTDESCR_DRPDWN_ID	Specifies the node description drop-down list box ID. PTDESCR_DRPDWN_ID is a key field.
PTORGDRPLSTITM_ID	The drop down list item ID for the specified drop down list. PTORGDRPLSTITM_ID is a key field.
PTND_DISPLAY_ORDER	Set the display order of the list item within the same dropdown box. By default, the list will display in the order the list items appear in the rowset.
PTNODELISTDESCR	The text that will appear for the drop down list item. The maximum length is 60 characters.

Field	Description
PTORGLISTDESCSTYLE	Specify the style class for the list item. The default is PT_ACTION_LIST_ITEM.

This table describes the fields in PTORGCRMB_SBR:

Field	Description
PTORGCRMBID	Assigns the breadcrumb ID. PTORGCRMBID is a key field.
PTPARENT_ORGCRMBID	Specifies the breadcrumb ID of Parent node. Only one row, which will be the first breadcrumb, can have this field blank or there will be a runtime error.
PTORGCRMBDESCR	Sets the breadcrumb description. In an organization chart this is generally a name.
DISPLAYED_FLAG	The displayed flag, Y = Displayed. N = Not displayed. The default is *"N"*.
PTORGCRMBLINKABLE	The linkable flag. Y = Linkable N = Not linkable The default is *"Y"*.

Minimum Methods

At a minimum, these methods are needed to create an organization chart:

- SetNodeData
- SetNodeRecord

These methods are required if your organization chart uses pop-ups:

- SetPopUpNodeData
- SetPopUpNodeRecord

These methods are required if your organization chart uses breadcrumbs:

- SetCrumbData
- SetCrumbRecord

These methods are required if your organization chart uses drop-down list boxes:

- SetDropdownData
- SetDropdownRecord

Data Type of an OrgChart Object

Chart objects are declared using the Chart data type. For example,

```
Local OrgChart &MyOrgChart;
```

Scope of an OrgChart Object

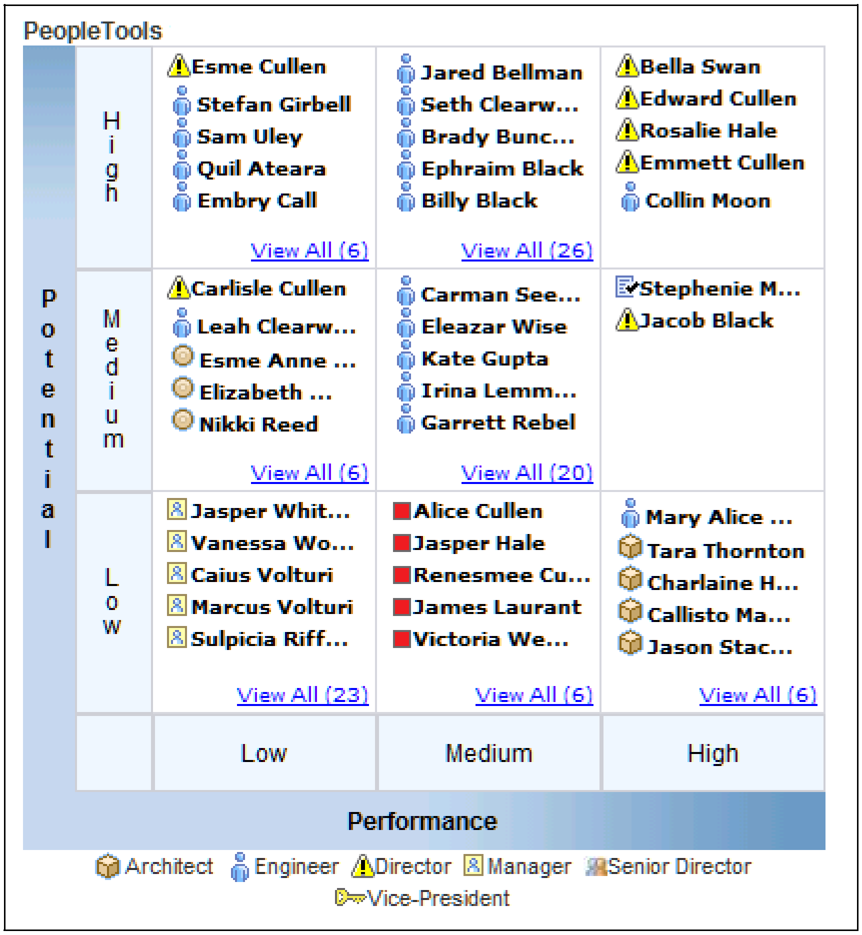
A chart object can be instantiated only from PeopleCode.

You can use this object only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message notification, a Component Interface, and so on.

Using the RatingBoxChart Class

A rating box chart is an interactive chart that displays nodes on a two-dimensional grid. A user can move the nodes from one box to another on the grid. When the user moves a node the underlying data updates automatically.

The rating box chart in this example displays ratings on the dimensions of potential and performance:



Rating box chart used to graph performance and potential

This section provides an overview of rating box chart terminology and discusses how to:

- Use rating box charts in PeopleSoft Pure Internet Architecture.
- Create rating box charts using the RatingBoxChart class.
- Rating box chart subrecords.
- Minimum methods.
- Minimum properties.

Understanding Rating Box Chart Terms

The following is a list of terms that apply specifically to rating box charts.

Box	A rating box chart is an array of boxes arranged in a grid along the X and Y axes. Each box represents a pair of ratings, one for the X value and one for the Y value.

Description	A text string that is associated with a node. The node description optionally displays with the node icon.
Icon	An image associated with a node that displays in a box on the rating box chart.
Node	A single data point on a rating box chart. A node represents an individual entity, such as an employee, customer, or product. A user can drag and drop a node to other boxes in the rating box chart grid.
Node Record	A record that contains the properties and data values for each node on the rating box chart. The node record must include a copy of the subrecord PTRATINGBIX_SBR.
Rating	A text string that is assigned to each box in the rating chart. Each box has an X rating, based on its position on the horizontal axis, and a Y rating, based on its position on the vertical axis.
Title	There are three titles: Main title, X-Axis title, and Y-Axis title. A title is text that identifies that portion of the chart.

Using Rating Box Charts in PeopleSoft Pure Internet Architecture

A rating box chart displays nodes on a chart that is divided into two or more boxes arrayed in a grid.

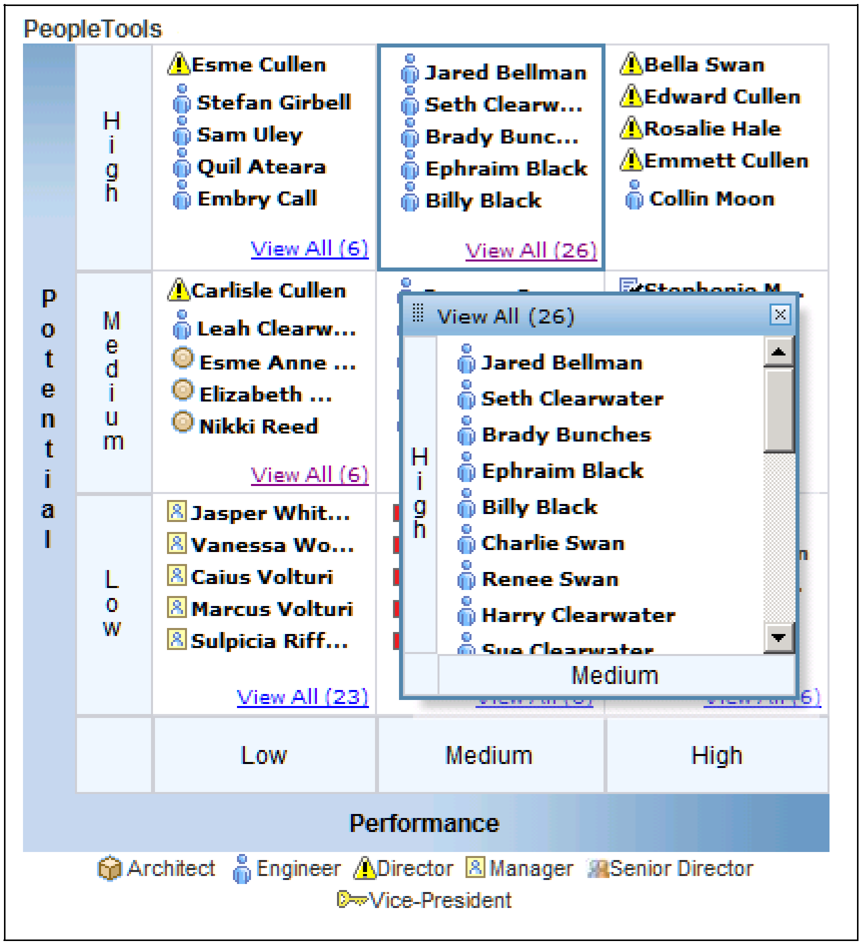
An application can correlate nodes, represented by icons, with various entities, such as employees, customers, or products, and various attribute pairs, such as potential/performance, cost/benefit, or opportunity/risk.

A user interacts with the rating box by dragging nodes to different boxes. When a node is moved, its underlying X (horizontal) and Y (vertical) values are updated. A user can also interact with the rating box by changing the node X/Y values in the grid that contains the rating box values if the application makes the grid visible and editable.

If a box contains more nodes than the number set in the RatingBoxChart property BoxMaxDisplayItems, the user can click a link at the bottom of the box to launch a pop-up that contains all the nodes. The box border changes color to indicate that the box has a pop-up open. A vertical scrollbar enables the user to scroll through all the nodes in the list.

The View all pop-up automatically closes if the user moves a node to another quadrant.

Using PeopleCode, you can associate each node with an image, or icon, and a description. You can configure a rating box to display nodes with icons and descriptions or with only icons.



Rating box chart with a pop-up displaying all nodes for a rating box

See Also

Chapter 11, "Charting Classes," BoxMaxDisplayItems, page 547

Creating Rating Box Charts Using the RatingBoxChart Class

When you add a rating box chart to a page in PeopleSoft Application Designer, you use the chart properties to associate the chart control with a record field. You will use this field name to identify the chart in PeopleCode.

You will need to create a record that will be the rating box chart node record. The record can be a standard database record or a derived/work record. The node record holds the values for the properties and data for each of the rating box chart nodes.

The record can have any name, but it must include a clone of the PeopleTools-delivered subrecord PTRATINGBOX_SBR. Since the subrecord will carry the FieldChange PeopleCode for the chart, in most cases you should create a unique subrecord for each chart. The node record must be placed on a page at level one on the component so that its PeopleCode will be loaded into the component buffer. It can be hidden.

In your chart PeopleCode, typically in the Activate event, you will create and populate a rowset linked to the chart's node record. This must be a component rowset, linked to the chart's node record, not a standalone rowset. The rowset has one row for each node in the chart.

Actions and Events

When user drags the node from one box to another, PeopleSoft Charting updates the values for the node's PTXAXISRATINGS and PTYAXISRATINGS values, changing the box's X and Y values to the new values. This change invokes field change processing on the PTXAXISRATINGS and PTYAXISRATINGS fields on the rating box node record. Typically, FieldChange PeopleCode performs business logic and refreshes the chart display.

Text Orientation

The orientation of the main title in a rating box chart is always horizontal.

The orientation of the X-axis title and X-axis labels is always horizontal.

The orientation of the Y-axis title and Y-axis labels is always vertical.

Rating Box Chart Subrecord Definition

These are the fields in PTRATINGBOX_SBR:

Field	***Description***
PTCHART_NODE	Specifies the node name. This is the key to the record.
PTND_DISPLAY_ORDER	Specifies the display order of the nodes within the chart. By default nodes will display in the same order as in the rowset. You must provide a unique value for display order for each node in the rating box rowset so that the nodes will display in the correct order in the rating box. A runtime error message is thrown when the display order is not unique for each node in the rating box rowset.
PTNODE_DESCR1	Specifies the main description for the node.
PTNODEDESCRSTYLE	Specifies the style class that defines the description's text font, type, color, size, and so on.
PTNDDESC1LINKABLE	Reserved for future use.
PTDESCR1_ICON_IMG	Specifies the image name for the main description.
PTDESCR1_ICON_POS	Indicates where to place the icon relative to the description. L – Left of the description R – Right of the description

Field	Description
PTXAXISRATINGS	Specifies the X-axis rating for the current node.
PTYAXISRATINGS	Specifies the Y-axis rating for the current node.
DISPLAYED_FLAG	Indicates whether the node will display in the chart area. Y – The node will display in the chart area. N – The node will not display in the chart area. The default value is *"Y"*.

Minimum Methods

At a minimum these methods are needed to create a rating box chart.

- SetRBNodeData
- SetRBNodeRecord
- SetXAxisLabels
- SetYAxisLabels

Minimum Properties

At a minimum, these properties are needed to create a rating box chart.

- XAxisBoxNum
- YAxisBoxNum

These properties are required if the chart includes a view all pop-up:

- PopUpWidth
- PopUpHeight

Data Type of a RatingBoxChart Object

Chart objects are declared using the Chart data type. For example,

```
Local RatingBoxChart &RatingBoxChart;
```

Scope of a RatingBoxChart Object

A chart object can be instantiated only from PeopleCode.

You can use this object only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message notification, a Component Interface, and so on.

Charting Classes Reference

The Charting Classes Reference contains these topics:

- Charting Classes Built-in Functions.
- Chart Class Methods.
- Chart Class Properties.
- Gantt Class Methods.
- Gantt Class Properties.
- OrgChart Class Methods.
- OrgChart Class Properties.
- RatingBoxChart Class Methods.
- RatingBoxChart Class Properties.
- Creating a Bar Chart Using the Chart Class.
- Creating a Gantt Chart.
- Creating an Organization Chart.
- Creating a Rating Box Chart.
- Creating a Chart Using an iScript.

Charting Classes Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateObject

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetChart

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetChartURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetGanttChart

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetOrgChart

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRatingBoxChart

Chart Class Methods

These methods are used by the Chart class. The methods are listed in alphabetical order.

Refresh

Syntax

`Refresh``()`

Description

Note.

This method has been deprecated and remains for backward compatibility only. Use the SetData method or SetOLData method instead.

If you make a change to the underlying data of a chart, call the SetData method again to update the chart.

You don't need to refresh after setting a property or method of the chart itself, such as setting the starting point or changing colors. When a method or property is used, the chart is automatically refreshed.

Parameters

None.

Returns

None.

Example

```
&MyChart.Refresh();
```

See Also

Chapter 11, "Charting Classes," SetData, page 435 and Chapter 11, "Charting Classes," SetOLData, page 447

Reset

Syntax

Reset()

Description

Use the Reset method to clear all existing chart settings and data.

Parameters

None.

Returns

None.

See Also

Chapter 11, "Charting Classes," Refresh, page 432

SetColorArray

Syntax

SetColorArray*(&Array_of_Color)*

Description

Use the SetColorArray method to set the colors for a series. You can specify only a one-dimensional array for the parameter. This method applies only to base chart data, not to the overlay data.

The type of array can be any of the following:

- String
- Number
- Integer
- Decimal

Use the SetColorArray method for line charts instead of SetDataColor because SetDataColor does not work with line charts. Individual line segments cannot be colored because there are fewer line segments than points. Use SetColorArray instead to color each complete line (series).

Parameters

Parameter	***Description***
&Array_of_Color	Specify an already instantiated array that contains the color values that you want for the series. You can specify either a constant or a number.

The following lists all the values you can use with the chart color methods. You can use either the numeric or constant value.

Numeric Value	***Constant Value***	***Description***
-1	%ChartColor_Series_Default	Default series color. When a data point is set to this, the default series color or the color that has been set by the user for that series is used instead. This value works only with the SetColorArray method.
0	%ChartColor_Black	Black
1	%ChartColor_Blue	Blue
2	%ChartColor_Cyan	Cyan
3	%ChartColor_DarkGray	DarkGray
4	%ChartColor_Gray	Gray
5	%ChartColor_Green	Green
6	%ChartColor_LightGray	LightGray
7	%ChartColor_Magenta	Magenta
8	%ChartColor_Orange	Orange
9	%ChartColor_Pink	Pink

Numeric Value	*Constant Value*	*Description*
10	%ChartColor_Red	Red
11	%ChartColor_White	White
12	%ChartColor_Yellow	Yellow
13	%ChartColor_Red_Orange	Red_Orange
14	%ChartColor_Yellow_Green	Yellow_Green
15	%ChartColor_Blue_Violet	Blue_Violet
16	%ChartColor_Purple	Purple
17	%ChartColor_Yellow_Orange	Yellow_Orange

Returns

None.

Example

The following example sets the four series in the chart:

```
&NumArray = CreateArray(1, 3, 6, 9);

&oChart.SetColorArray(&NumArray);
```

See Also

Chapter 8, "Array Class," page 257

SetData

Syntax

SetData({*Record_Name* | *&Rowset*})

Description

Use the SetData method to specify where the data from the chart is coming from.

You can use either a record name or an already instantiated, populated rowset. In general, specify a record when building a chart from page data and a rowset when building a chart from a standalone rowset.

If you make a change to the underlying data of a chart, call the SetData method again to update the chart.

Parameters

Parameter	***Description***
Record_Name	Specify the name of a record to be used to populate the chart with data.
&Rowset	Specify the name of an already instantiated rowset to populate the chart with data. This is generally a standalone rowset.

Returns

None.

Example

The following is the minimum code you need to create a PeopleSoft chart.

```
Local Chart &MyChart;

&MyChart = GetChart(ABS_HIST.CHART);
&MyChart.SetData(ABS_HIST);
&MyChart.SetDataYAxis(ABS_HIST.Reason);
&MyChart.SetDataXAxis(ABS_HIST.Duration);
```

The following example creates a chart using a standalone rowset.


```
Function IScript_GetChartURL()
local object &MYCHART;
local string &MYURL;
local rowset &MYROWSET;

&MYCHART = CreateObject("Chart");

&MYROWSET = CreateRowset(Record.ABS_HIST);

&EmplID = %Emplid;

&MYROWSET.Fill("Where EMPLID :=1", &EmplID);

&MYCHART.SetData(&MYROWSET);
&MYCHART.SetDataXAxis(ABS_HIST.ABSENCE_TYPE);
&MYCHART.SetDataYAxis(ABS_HIST.DURATION_DAYS);

&MYURL = %Response.GetChartURL(&MYCHART);

/* use &MYURL in your application */
...
End-Function;
```

See Also

Chapter 11, "Charting Classes," SetDataXAxis, page 444 and Chapter 11, "Charting Classes," SetDataYAxis, page 446

SetDataAnnotations

Syntax

SetDataAnnotations*(Record_Name.Field_Name)*

Description

Use the SetDataAnnotations method to specify an optional text label that can be associated with each point in the chart.

Parameters

Parameter	***Description***
Recordname.Fieldname	Specify the field name (and the record it's associated with) that contains the text for the optional label.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetOLDataAnnotations, page 448

SetDataColor

Syntax

`SetDataColor``(Record_Name.Field_Name)`

Description

Use the SetDataColor method to specify a field that contains the color for each data point. Each field for each data point must contain either a number or a constant that specifies the color you want to use.

Use the SetColorArray method for line charts instead of SetDataColor because SetDataColor does not work with line charts. Individual line segments cannot be colored because fewer line segments exist than points. Use SetColorArray instead to color each complete line (series).

Note. This method has no effect when used with an overlay.

Parameters

Parameter	***Description***
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the colors for each data point. You can specify either a constant or a number.

See Chapter 11, "Charting Classes," SetColorArray, page 433.

Returns

None.

SetDataGlyphScale

Syntax

`SetDataGlyphScale``(Record_Name.Field_Name)`

Description

Use the SetDataGlyphScale method to specify a field that contains numerical data defining the size of each glyph for bubble charts (scatter charts). These values are mapped to a range of size values between a predefined minimum and maximum.

This method only affects scatter charts (%ChartType_2DScatter)

Parameters

Parameter	***Description***
Recordname.Fieldname	Specify the field (and the record associated with it) that contains numerical data used to define the size of each glyph.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetOLDataGlyphScale, page 449

SetDataHints

Syntax

`SetDataHints``(`*`Record_Name.Field_Name`*`)`

Description

Use the SetDataHints method to specify the field that contains the text that displays as a data hint.

What displays for a data hint depends on the value specified for the data hints method, as well as the value specified for the series and the data method.

Series Data but No Data Hints

If you do not specify data hints, but you *do* specify both the data series and data methods, the hints appear as follows:

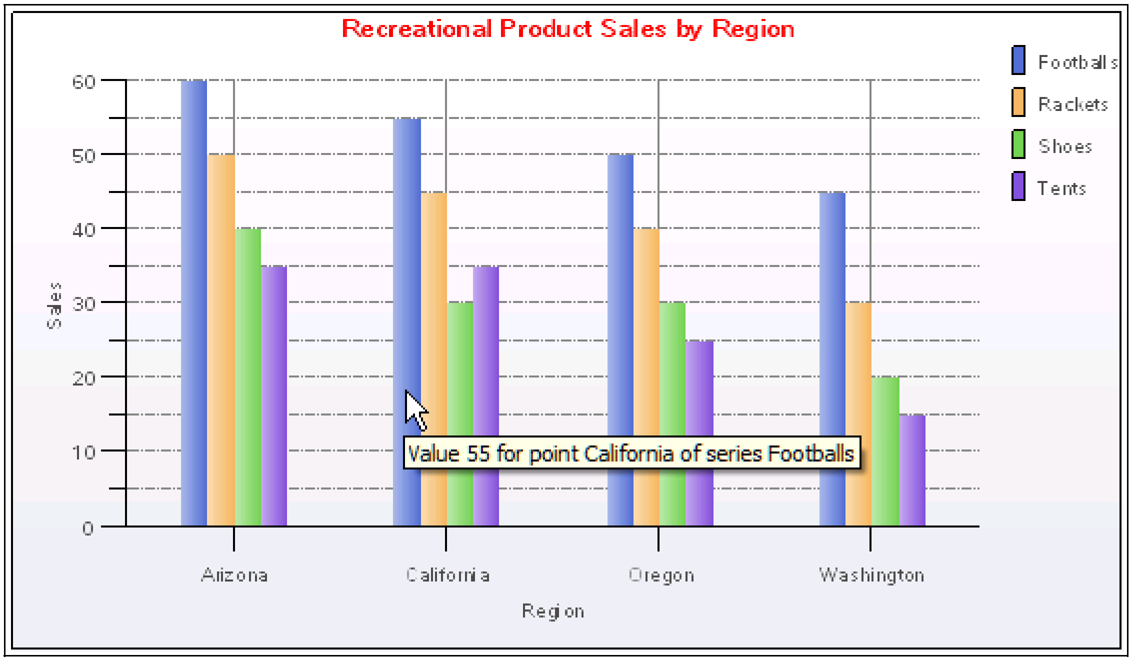
`Value` *`value`* `for point` *`x position`* `of series` *`series`*

Where:

- *value* is the Y data value

- *x position* is the X-axis label
- *series* is the series value

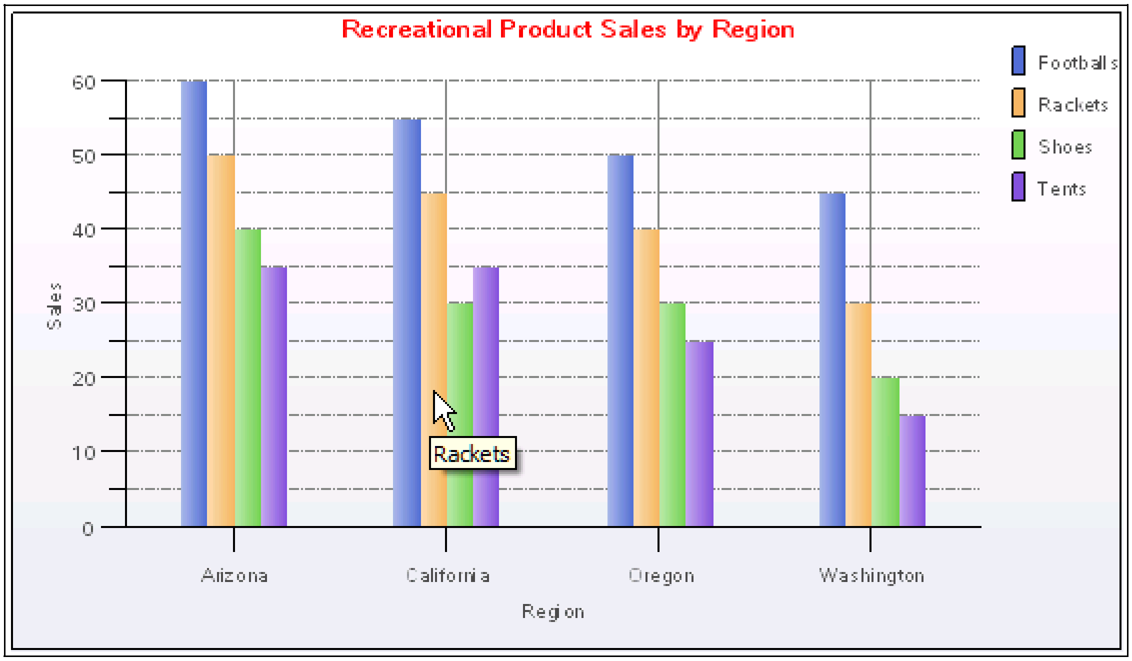
The following example shows this type of data hints. The tooltips pop-up appears when the cursor hovers over any data value.



Data hints with multiple series example

Data Hints, Data, and Series

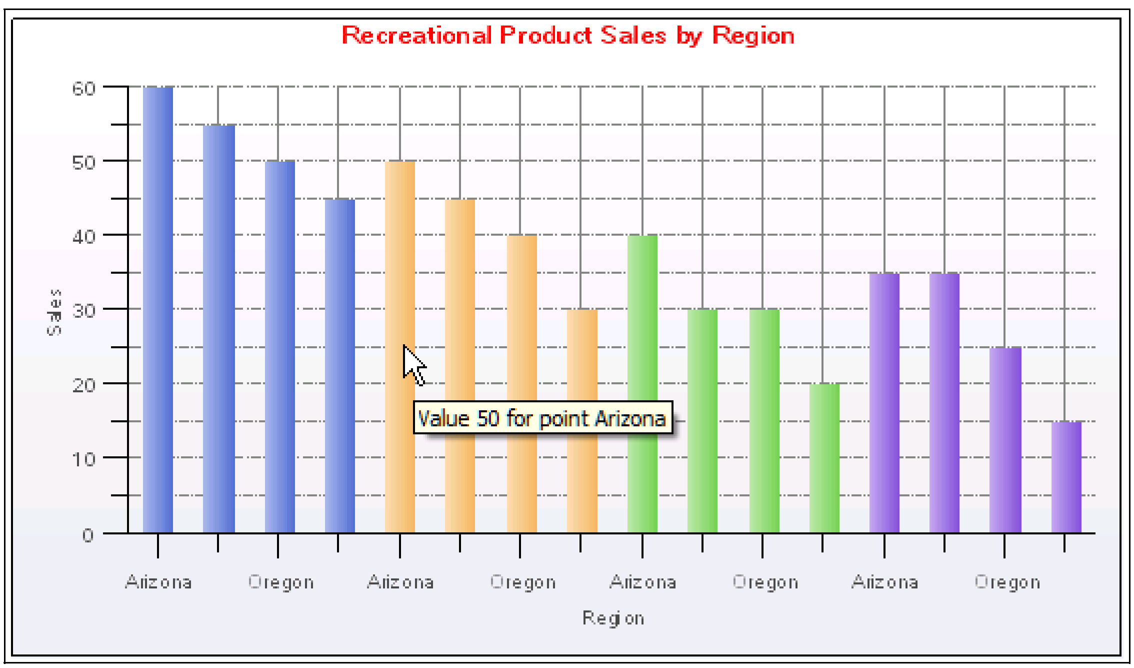
If you specify data hints, then that value appears. In the following example, the data hints are set to the same value as the series is set, that is, the product:



Data hints set to the product value

Data and Series, but no Data Hints

If you do not specify SetDataHints and do not specify SetDataSeries, then the value of the Y axis and the value of the X axis appear in the data hint:



Data hints from default data

Parameters

Parameter	*Description*
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data hints for the chart.

Returns

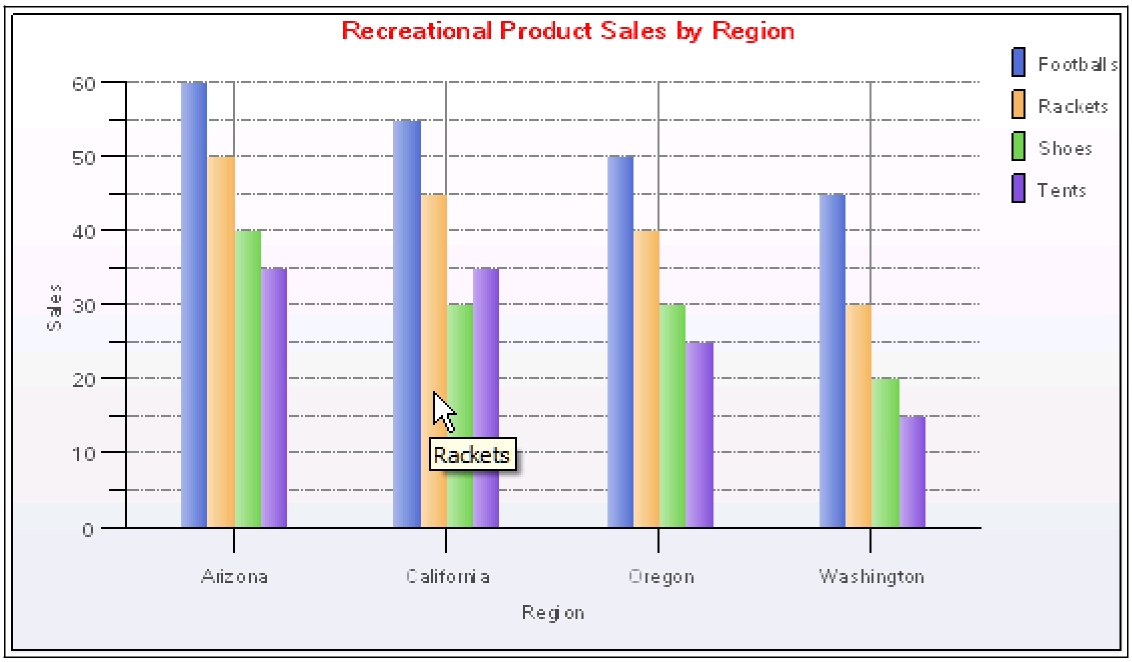
None.

Example

The following PeopleCode example:

```
&oChart.SetDataHints(QE_CHART_RECORD.QE_CHART_REGION);
```

Could produce data hints as follows:



Data hints set to the product value

See Also

Chapter 11, "Charting Classes," SetDataXAxis, page 444 and Chapter 11, "Charting Classes," SetDataSeries, page 443

SetDataSeries

Syntax

SetDataSeries*(Record_Name.Field_Name)*

Description

Use the SetDataSeries method to specify the name of the field containing the series values. Every distinct value in this field is considered a unique series.

The series is plotted in the order given by the record.

By default, the legend is populated by the value of this field.

If this value is Null, the system assumes there is only one series to plot.

Parameters

Parameter	*Description*
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the series values for the chart.

Returns

None.

Example

```
&MYCHART.SetDataSeries(MYRECORD.MYSERIES);
```

See Also

Chapter 11, "Charting Classes," SetDataXAxis, page 444 and Chapter 11, "Charting Classes," SetDataYAxis, page 446

SetDataURLs

Syntax

SetDataURLs*(FieldName)*

Description

Use the SetDataURLs method to specify a URL to be launched when a data point is clicked in the chart. The URL for each point must be given as a string, that is, enclosed in quotation marks. You can also specify *recordname.fieldname.* Use this method for charts created from a chart control or a standalone rowset.

You could use this method to launch a JavaScript that generates a pop-up menu.

Parameters

Parameter	*Description*
FieldName	Specify a field name to be used to populate the URL for each data point.

Returns

None.

Example

```
&MyChart.SetDataURLs(MyRecord.URL);
```

SetDataXAxis

Syntax

`SetDataXAxis`*`(Record_Name.Field_Name)`*

Description

Use the SetDataXAxis method to specify the groups along the X axis.

If this value is not set or Null, the Y values are plotted along the X axis in groups labeled by their order number.

The order of the data is plotted in the order of the data in the record.

By default, the labels along the X axis are populated by the value of this field.

Parameters

Parameter	Description
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data for the X axis for the chart.

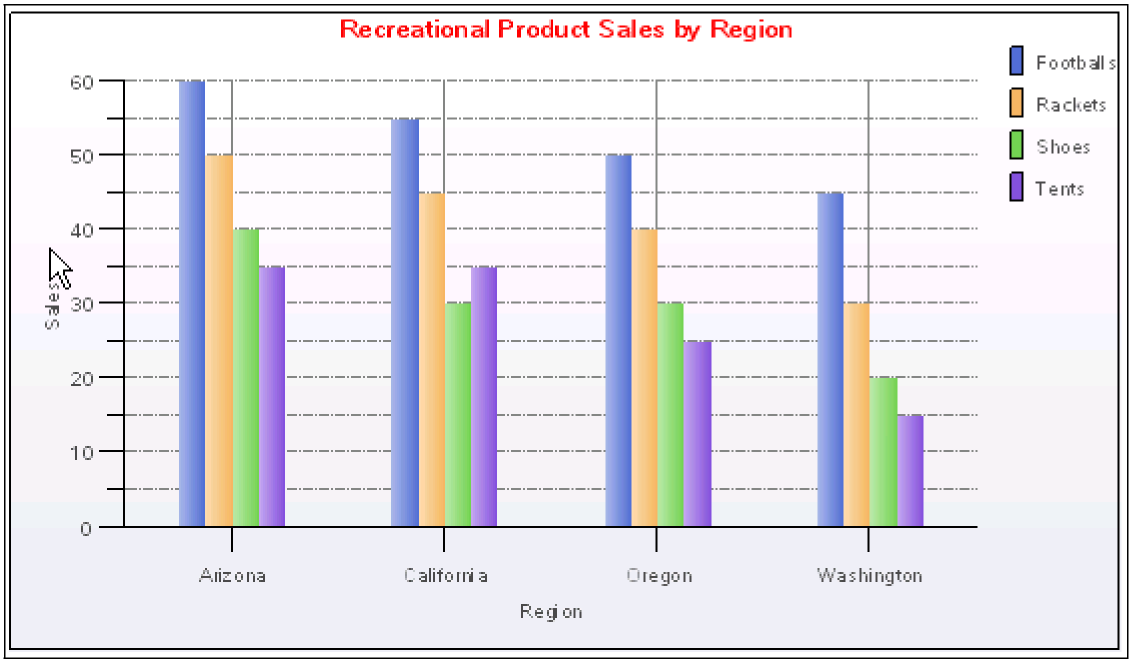
Returns

None.

Example

In the following example, the Y axis is set to a numeric amount, while the X axis is set to the region.

```
&cChart = GetChart(PB_CHART_DUMREC.PB_CHART_FIELD);

&cChart.SetData(Record.PB_CHART_RECORD);
&cChart.SetDataSeries(PB_CHART_RECORD.PB_CHART_PRODUCT);
&cChart.SetDataXAxis(PB_CHART_RECORD.PB_CHART_REGION);
&cChart.SetDataYAxis(PB_CHART_RECORD.PB_CHART_SALES);
```



X axis set with region data

See Also

Chapter 11, "Charting Classes," SetDataYAxis, page 446 and Chapter 11, "Charting Classes," SetData, page 435

SetDataYAxis

Syntax

`SetDataYAxis``(`*`Record_Name.Field_Name`*`)`

Description

Use the SetDataYAxis to specify the data to plot in the chart.

The Y axis is always considered the data axis.

The Y axis must always be numeric. If the greatest Y-axis value is less than 10, the Y-axis labels are given a decimal place. If the greatest Y-axis value is greater than or equal to 10, the Y-axis labels appear as integers.

Note. If you have specified negative numbers in the Y axis, then the X axis is drawn according to the XAxisCross property.

Parameters

Parameter	*Description*
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data for the Y axis for the chart.

Returns

None.

Example

```
&oChart.SetDataYAxis(QE_CHART_RECORD.QE_CHART_SALES);
```

See Also

Chapter 11, "Charting Classes," SetDataXAxis, page 444 and Chapter 11, "Charting Classes," SetData, page 435

SetLegend

Syntax

`SetLegend``(`*`&Array_of_String`*`)`

Description

Use the SetLegend method to specify alternative legends. By default, the legends are populated from the record field specified with SetDataSeries.

This method is used to write both the overlay legend and the series legend. The labels are overwritten in the order of elements in the array, that is, the first element overwrites the first series, the second overwrites the second, and so on, with the last element in the array overwriting the overlay legend. There can be more than one series in an overlay.

If you do not specify an element for an array (that is, a blank or a null) then no legend is listed for that series.

Note. Default label text is not automatically translated. If you set your own labels, be sure to use translated text, such as message catalog entries.

Parameters

Parameter	***Description***
&Array_of_String	Specify an already instantiated array of string, containing the text that you want to use for the legend.

Returns

None.

Example

The following example displays a legend containing only the overlay.

```
&LegendArray = CreateArray("", "", "Revenue");
```

```
&MyChart.SetLegend(&LegendArray);
```

See Also

Chapter 11, "Charting Classes," SetDataSeries, page 443

Chapter 8, "Array Class," page 257

SetOLData

Syntax

`SetOLData``({`*`Record_Name`*` | `*`&Rowset`*`})`

Description

Use the SetOLData method to specify where the data for the overlay is coming from.

If you make a change to the underlying data of a chart, call the SetOLData method again to update the chart.

There are no default values for the overlay data. You must specify the data.

You can specify two different Y-axis fields, but you must make certain that different data plotted against the same axis makes sense. For example, while plotting two currency amounts would work technically—plotting currency amounts against number of units sold and sharing the same axis— would not look correct.

Parameters

Parameter	***Description***
Record_Name	Specify the name of a record to be used to populate the overlay of the chart with data.
&Rowset	Specify the name of an already instantiated rowset to populate the overlay of the chart with data.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDataXAxis, page 444; Chapter 11, "Charting Classes," SetDataYAxis, page 446 and Chapter 11, "Charting Classes," SetData, page 435

SetOLDataAnnotations

Syntax

`SetOLDataAnnotation``(Record_Name.Field_Name)`

Description

Use the SetOLDataAnnotations method to specify an optional text label that can be associated with each point in the overlay of the chart.

Parameters

Parameter	*Description*
Recordname.Fieldname	Specify the field name (and the record it's associated with) that contains the text for the optional label.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDataAnnotations, page 437

SetOLDataGlyphScale

Syntax

`SetOLDataGlyphScale``(`*`Record_Name.Field_Name`*`)`

Description

Use the SetOLDataGlyphScale method to specify a field that contains numerical data defining the size of each glyph in the overlay chart. These values are mapped to a range of size values between a predefined minimum and maximum.

Parameters

Parameter	*Description*
Recordname.Fieldname	Specify the field (and the record associated with it) that contains numerical data used to define the size of each glyph.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDataGlyphScale, page 438

SetOLDataSeries

Syntax

SetOLDataSeries*(Record_Name.Field_Name)*

Description

Use the SetOLDataSeries method to specify the name of the field containing the overlay series values. Every distinct value in this field is considered a unique series.

The series is plotted in the order given by the rows in the record.

If this value is Null, the system assumes there is only one series to plot.

Parameters

Parameter	***Description***
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the series values of the overlay for the chart.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDataSeries, page 443

SetOLDataXAxis

Syntax

SetOLDataXAxis*(Record_Name.Field_Name)*

Description

Use the SetOLDataXAxis method to specify the groups along the X axis if the axis is non-numeric for the overlay.

If the X axis is numeric, this method is used to give the position of the points along the axis. Only discrete values are supported for the X axis.

If this value is Null, the Y values are plotted along the X axis in groups labeled by their order number.

The order of the data is plotted in the order of rows in the record.

Parameters

Parameter	***Description***
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data for the X axis for the overlay of the chart.

Returns

None.

SetOLDataYAxis

Syntax

SetOLDataYAxis*(Record_Name.Field_Name)*

Description

Use the SetOLDataYAxis to specify the data to plot for the overlay of the chart.

This value must always be numeric.

This method has no default value.

Parameters

Parameter	***Description***
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data for the overlay of the chart.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDataXAxis, page 444 and Chapter 11, "Charting Classes," SetDataYAxis, page 446

SetXAxisLabels

Syntax

SetXAxisLabels*(&Array_of_Any)*

Description

Use the SetXAxisLabels method to specify an array of labels for the X axis. By default, the labels are populated from the record field specified by SetDataXAxis. Labels specified with SetXAxisLabels overwrite the default labels.

Note. Default label text is not automatically translated. If you set your own labels, be sure to use translated text, such as message catalog entries.

Parameters

Parameter	*Description*
&Array_of_Any	Specify an already instantiated array of any that contain the labels that you want to use for the X Axis.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDataXAxis, page 444

Chapter 8, "Array Class," page 257

Chapter 11, "Charting Classes," Tick Mark Considerations, page 584

SetYAxisLabels

Syntax

SetYAxisLabels*(&Array_of_Any)*

Description

Use the SetYAxisLabels method to specify an array of labels for the Y axis. Labels specified with SetYAxisLabels overwrite the default labels.

If the greatest Y-axis value is less than ten, the default Y-axis labels are given a decimal place. If the greatest Y-axis value is greater than or equal to ten, the default Y-axis labels are displayed as integers.

Note. Default label text is not automatically translated. If you set your own labels, be sure to use translated text, such as message catalog entries.

If you set the labels yourself, you may need to set the YAxisTicks and YAxisMax because you no longer have numbers that correlate to the data points.

Parameters

Parameter	***Description***
&Array_of_Any	Specify an already instantiated array of any that contain the labels that you want to use for the Y Axis.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDataXAxis, page 444; Chapter 11, "Charting Classes," YAxisTicks, page 475 and Chapter 11, "Charting Classes," YAxisMax, page 473

Chapter 8, "Array Class," page 257

Chapter 11, "Charting Classes," Tick Mark Considerations, page 584

Chart Class Properties

These properties are used by the Chart class. The properties are described in alphabetic order.

DataStartRow

Description

This property sets or returns the row number (after any preprocessing) at which to start plotting. This is useful when you have many rows of data in a rowset, and you want to start at a particular row. This can also be useful for creating your own 'scrolling' effect, by specifying both the start row and how many rows to be displayed (by using the DataWidth property).

If this property is not set, then the value is 1.

This property is read-write.

See Also

Chapter 11, "Charting Classes," DataWidth, page 454

DataWidth

Description

This property returns or sets the number of rows to plot for a series. The starting point for the number of rows is set with the DataStartRow property. If the DataWidth property value is not set, then the width is from the DataStartRow to the last element in the rowset. If the property value is less than zero, then it will automatically be set to one.

This property is read-write.

Example

Suppose your rowset returned 400 rows. You wouldn't want all 400 to display in your chart. Instead, you'd pick a subset of those rows. The row to start with is set with DataStartRow, while how many rows to display is set with DataWidth.

The following code could be used with a push button connected with a chart. The push button enables the next set of data to appear with the chart.

You do not need to refresh after setting this property. When a method or property is used, the chart is automatically refreshed.

```
Local Chart &MyChart;

&MyChart = GetChart(ChartRec.DisplayField);
&Start = &MyChart.DataStartRow;
/* display the next 20 rows of data */
&MyChart.DataStartRow = &Start + 20;
&MyChart.DataWidth = 20;
```


See Also

Chapter 11, "Charting Classes," DataStartRow, page 454

GridLines

Description

Use this property to specify whether to show grid lines with the chart.

Note. This property is ignored unless you have set RevertToPre850 to True for the chart.

If you do not specify a value, both horizontal and vertical grid lines are shown.

This property controls only the display of grid lines. If you want to specify the style of the grid lines, use the GridLineType property.

The values are:

Numeric Value	***Constant Value***	***Description***
0	%ChartGrid_None	Don't show grid lines
1	%ChartGrid_Horizontal	Show horizontal grid lines (default)
2	%ChartGrid_Vertical	Show vertical grid lines
3	%ChartGrid_Both	Show both vertical and horizontal grid lines

This property is read-write.

See Also

Chapter 11, "Charting Classes," GridLineType, page 455

GridLineType

Description

Use this property to specify the style of the grid lines when either or both vertical or horizontal grid lines are turned on.

Note. This property is ignored unless you have set RevertToPre850 to True for the chart.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

The default value is %ChartLine_Solid. The possible values are:

Value	***Description***
%ChartLine_Solid	Draw grid lines with solid lines.
%ChartLine_Dash	Draw grid lines with lines composed of dashes.
%ChartLine_Dot	Draw grid lines with lines composed of dots.
%ChartLine_DashedDot	Draw grid lines with lines composed of both dashes and dots.

This property is read-write.

HasLegend

Description

Use this property to specify if a legend is displayed with the chart. This property takes a Boolean value: True, if the legend is displayed with the chart, False otherwise.

The default value is False.

This property is read-write.

Height

Description

Use this property to specify the height of the chart. This property takes a numeric value. The unit of measurement is pixels.

Generally this property is used with charts created for iScripts, to specify the height of the image, before generating the image map. You can also use it to overwrite the height set in PeopleSoft Application Designer.

If you try to read this property before setting it, the value returned is zero.

Note. The Height and Width properties must be set to the same value to change the size of a 2D Scatter chart, that is, a chart with the Type property set to %ChartType_2DScatter.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Height, page 456

ImageMap

Description

This property returns the HTML client-side image map. The image map is what makes the chart interactive.

This property is read-only.

See Also

Chapter 11, "Charting Classes," Creating a Chart Using an iScript, page 596

IsDrillable

Description

Use this property to specify whether the end-user can click on the chart and trigger a PeopleCode program in the FieldChange event for that row.

By default, when the chart data comes from a record, a PeopleCode program in the FieldChange event for the Y-Data field of the clicked row is executed.

Note. The record that contains the PeopleCode must be in the component buffer at runtime.

See Chapter 11, "Charting Classes," Creating a Chart Using the Chart Class, page 557.

This property takes a Boolean value: True if the end-user can click on the chart to initiate an action, False otherwise.

Note. Setting this property has no effect if the IsPlainImage property is set as True.

Note. When using a chart in an iScript, this property must be used in conjunction with the SetDataURLs method. The SetDataURLs method must point to a field that is populated with the URLs to be triggered for each data point in the chart.

The default value for this property is False.

This property is read-write.

See Also

Chapter 11, "Charting Classes," IsPlainImage, page 458

IsPlainImage

Description

Use this property to specify whether the chart is built without any extra HTML, such as SRC or MAP tags.

If this property is specified as True, the end-user will not be able to click on the chart to trigger an event. In addition, the pop-up data hints will not appear when the end-user passes a mouse over the chart. However, you may see a performance improvement if your chart has an extremely complicated image map and you specify this property as True.

This property takes a Boolean value: True if the chart is built without extra HTML, False otherwise.

The default value of this property is False.

This property is read-write.

IsTrueXY

Description

Use this property to enable a TrueXY, or numeric, axis. This causes the X-axis data to be loaded as continuous linear numeric data, rather than discrete data, as is used in bar charts for example.

This property is only applicable to scatter, line or histogram charts where the X axis can be numeric.

This property is read-write.

See Also

Chapter 11, "Charting Classes," True XY Line Chart, page 578

IsYAxisInteger

Description

Specify whether the Y axis will display integers. Set IsYAxisInteger to True if the Y axis represents integer data, such as number of people or number of tasks.

IsYAxisInteger is not used with pie charts and percentage bar charts.

This property takes a Boolean value.

The default value of this property is False.

This property is read-write.

LegendMaxEntries

Description

Use this property to specify the number of legend entries before a new column (or row) is created.

When the legend position is specified as left or right, the legend is drawn vertically so a new column is created when this property is exceeded.

When the legend position is specified as top or bottom, the legend is drawn horizontally and a new row is created when this property is exceeded.

The default value of this property is five.

This property is read-write.

LegendPosition

Description

Use this property to specify where the legend should appear, in relationship to the chart. You can specify either a numeric or constant value for this property.

Note. ChartLegend_Separate generates a legend without a chart. The legend takes over the entire charting area. The legend appears in the center of the chart.

The values are:

Numeric Value	***Constant Value***	***Description***
0	%ChartLegend_Left	Display legend to the left of the chart.
1	%ChartLegend_Right	Display legend to the right of the chart.
2	%ChartLegend_Top	Display legend on top of the chart.
3	%ChartLegend_Bottom	Display legend below the chart.
4	%ChartLegend_Separate	Generate a legend without a chart.

This property is read-write.

LegendStyle

Description

Use this property to specify the style of the legend. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

LineType

Description

Use the LineType property to specify the style of series that are plotted as lines. This is for a regular series only. To specify the style of line for overlay data, use the OLLineType property.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

This property takes either a numeric or constant value. The values are:

Numeric Value	***Constant Value***	***Description***
1	%ChartLine_Solid	Draw series with a solid line.
2	%ChartLine_Dash	Draw series with a line composed of dashes.
3	%ChartLine_Dot	Draw series with a line composed of dots.
4	%ChartLine_DashedDot	Draw series with a line composed of both dashes and dots.

This property is read-write.

See Also

Chapter 11, "Charting Classes," OLLineType, page 462

MainTitle

Description

Use this property to specify the text for the main title of the chart.

This property takes a string value.

This property is read-write.

MainTitleOrient

Description

Use this property to specify the orientation of the text for the main title.

The default value is horizontal.

You can specify either a number or a constant for this property. The values are:

Numeric Value	***Constant Value***	***Description***
0	%ChartText_Horizontal	Title text is displayed horizontally.
90	%ChartText_Vertical	Title text is displayed vertically.

This property is read-write.

MainTitleStyle

Description

Use this property to specify the style of the main title. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

OLLineType

Description

Use the OLLineType property to specify the style of an overlay line.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

This property takes either a numeric or constant value. The values are:

Numeric Value	***Constant Value***	***Description***
1	%ChartLine_Solid	Draw series with a solid line.
2	%ChartLine_Dash	Draw series with a line composed of dashes.
3	%ChartLine_Dot	Draw series with a line composed of dots.
4	%ChartLine_DashedDot	Draw series with a line composed of both dashes and dots.

This property is read-write.

See Also

Chapter 11, "Charting Classes," LineType, page 460

OLType

Description

Use this property to specify the style of the overlay line.

You can specify either a numeric or a constant value for this property. The values are:

Numeric Value	***Constant Value***	***Description***
4	%ChartType_2DLine	Overlay is drawn as a 2D line.
5	%ChartType_2DHistogram	Overlay is drawn as a 2D histogram.

This property is read-write.

RevertToPre850

Description

PeopleTools Version 8.50 implements new chart style classes that bring a consistent look and feel to charts.

Use this property to specify whether the chart will revert to the pre-PeopleTools 8.50 chart style classes.

Warning! By default RevertToPre850 is set to False, which means that charts that were created prior to PeopleTools 8.50 will use the new 8.50 chart style classes.

Any custom Chart class properties that were set in PeopleCode for existing charts that use the new 8.50 chart style classes will be ignored if those properties are defined in the 8.50 chart style classes.

Oracle strongly recommends that you review existing charts to verify that they display correctly.

This property takes a Boolean value:

Parameter	***Description***
True	Do not use the PeopleTools 8.50 style classes for this chart so that the chart will maintain its original appearance.
False	Use the PeopleTools 8.50 style classes for this chart. Any chart properties that are set in application PeopleCode may be overridden by the 8.50 style classes.

The default is False.

This property is read-write.

See Also

Chapter 11, "Charting Classes," OLLineType, page 462

RotationAngle

Description

Note. This property has been deprecated and remains for backward compatibility only. Use the XRotationAngle, YRotationAngle, and ZRotationAngle properties instead.

Use this property to specify the rotation angle of the chart. This property is valid only with 3D style charts, that is, charts that have Type set as %ChartType_3DBar, %ChartType_3DStackedBar, and so on.

Note. This property has no effect on the 3D pie chart.

You don't need to refresh after setting this property. When a method or property is used, the chart is automatically refreshed.

This property takes a numeric value from 1 to 360.

This property does not take negative numbers.

The chart is rotated in a clockwise direction.

This property is read-write.

ShowCrossHair

Description

Use the ShowCrossHair property to specify whether the chart should be split into quadrants.

This property takes a boolean value. If you specify true, two lines are drawn, one from the middle of the X axis and one from the middle of the Y axis.

This property is read-write.

Style

Description

A *style sheet* is a definition, like a record or field definition, that you create in PeopleSoft Application Designer. It contains a collection of formatting styles, called *classes*, each of which can be applied to either the entire chart or to specific elements within a chart.

Use this property to specify the overall style of the chart by specifying a style class. The value must be a valid class within the style sheet specified for the chart.

Individual style properties (such as SetXAxisStyle, MainTitleStyle, and so on) can be used to override this value for individual elements of the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

StyleSheet

Description

Use this property to associate a style sheet with a chart.

The PTSTYLEDEF style sheet is the default style sheet associated with a chart.

Both the Style property for the chart, and individual style properties (such as SetXAxisStyle, MainTitleStyle, and so on) can be used to override this value for individual elements of the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

Type

Description

Use this property to specify the type of chart that you want. You can specify either a numeric or constant value for this property. The valid values are:

Numeric Value	***Constant Value***	***Description***
0	%ChartType_2DBar	Two-dimensional bar chart
1	%ChartType_2DStackedBar	Two-dimensional stacked bar chart
2	%ChartType_2DPercentBar	Two-dimensional percent bar chart
3	%ChartType_2DHorizStackedBar	Two-dimensional stacked horizontal bar chart
4	%ChartType_2DLine	Two-dimensional line chart
5	%ChartType_2DHistogram	Two-dimensional histogram chart
6	%ChartType_2DPie	Two-dimensional pie chart
7	%ChartType_3DBar	Three-dimensional bar chart
8	%ChartType_3DStackedBar	Three-dimensional stacked bar chart
9	%ChartType_3DPie	Three-dimensional pie chart
10	%ChartType_2DHorizPercentBar	Two-dimensional horizontal percent chart
11	%ChartType_3DPercentBar	Three-dimensional percent bar char
13	%ChartType_2DHorizBar	Two-dimensional horizontal bar chart
14	%ChartType_Scatter	Scatter chart.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Chart Class Chart Types, page 396

Width

Description

Use this property to specify the width of the chart. This property takes a numeric value. The unit of measurement is pixels.

Generally this property is used with charts created for iScripts, to specify the width of the image, before generating the image map. You can also use it to overwrite the width set in PeopleSoft Application Designer.

If you try to read this property before setting it, the value returned is zero.

Note. The Height and Width properties must be set to the same value to change the size of a 2D Scatter chart, that is, a chart with the Type property set to %ChartType_2DScatter.

This property is read-write.

XAxisCross

Description

Use this property to specify the appearance of the X-axis data when there are both negative and positive values.

You can specify either a numeric value or a constant for this property. Values are:

Numeric Value	***Constant Value***	***Description***
0	%ChartXAxisCross_Zero	The X axis extrudes from the zero point on the Y axis. X-axis labels are located below the X axis.
–1	%ChartXAxisCross_Bottom	The X axis extrudes from the Y minimum on the Y axis.
–2	%ChartXAxisCross_ZeroLineBttm	The X axis extrudes from the Y minimum on the Y axis, and a line is drawn from the zero point on the Y axis.
-3	%ChartXAxisCross_ZeroLblsBttm	The X axis extrudes from the zero point on the Y axis. X-axis labels are located at the bottom of the chart.

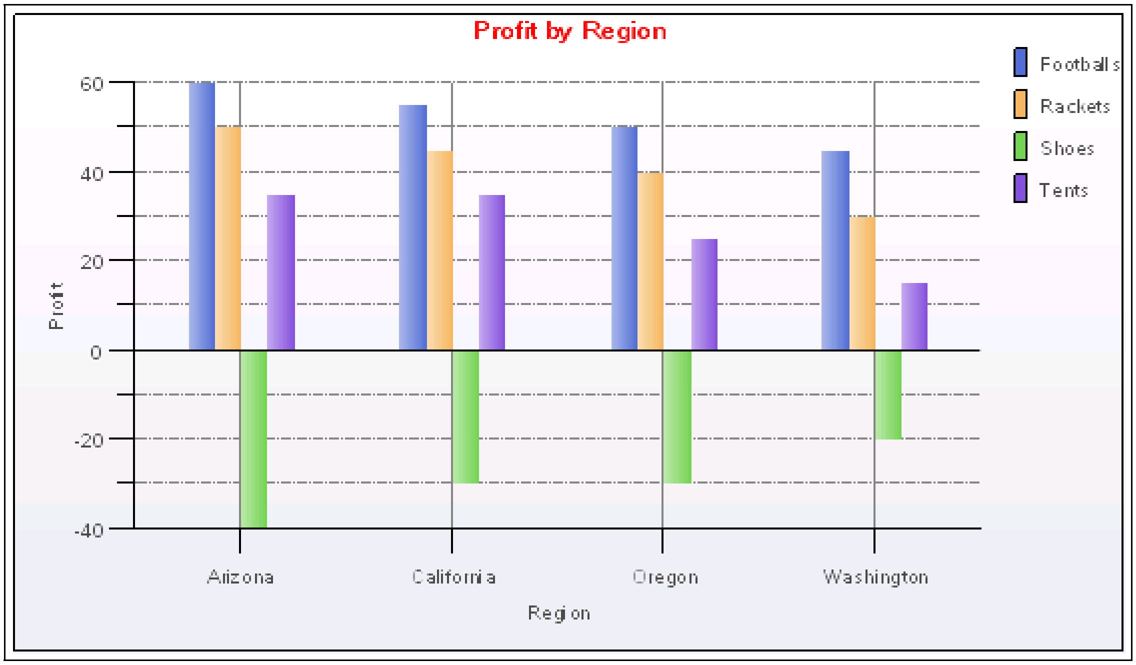
This property is read-write.

Example

The following code:

```
&MyChart.XAxisCross = %ChartXAxisCross_ZeroLineBttm;
```

produces a chart similar to this:



Example of %ChartXAxisCross_ZeroLineBttm

XAxisCrossPoint

Description

Use the XAxisCrossPoint property to specify the point at which the X axis intersects a numeric Y axis. The value is in the coordinate system of the Y-axis scale.

This property is read-write.

See Also

Chapter 11, "Charting Classes," YAxisCrossPoint, page 472

XAxisLabelOrient

Description

Use this property to specify whether the X-axis label is displayed horizontally or vertically.

The default value is horizontal.

You can specify either a numeric value or a constant for this property. Values are:

Numeric Value	***Constant Value***	***Description***
0	%ChartText_Horizontal	The X-axis label is displayed horizontally.
90	%ChartText_Vertical	The X-axis label is displayed vertically

This property is read-write.

XAxisMax

Description

The maximum value of the X axis.

This property is only applicable to scatter, line or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

This property is read-write.

XAxisMin

Description

The minimum value of the X axis.

This property is only applicable to scatter, line or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

This property is read-write.

XAxisPrecision

Description

Use the XAxisPrecision property to control the number of decimal places displayed in the labels of a numeric X axis.

This property is only applicable to scatter, line or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

This property is read-write.

See Also

Chapter 11, "Charting Classes," YAxisPrecision, page 473

XAxisScaleResolution

Description

Use the XAxisScaleResolution property to control the level of detail displayed on a numeric axis. Use this property to increase or reduce the number of major tick marks and labels on the X axis.

This property is only applicable to scatter, line, bubble, or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

The values are:

Numeric Value	*Constant Value*	*Description*
0	%ChartAxisResolution_Fine	Specify fine resolution for the chart, that is, increase the number of major ticks and labels on the X axis. The fine scale resolution uses extra tick marks to give a more detailed division of the axis. This means that there may not be a one-to-one relationship between the axis labels and the tick marks.
1	%ChartAxisResolution_Coarse	Specify coarse resolution for the chart, that is, reduce the number of major ticks and labels on the X axis. The coarse scale value maintains a one-to-one relationship between the axis labels and the tick marks.

This property is read-write.

See Also

Chapter 11, "Charting Classes," YAxisScaleResolution, page 474

XAxisStyle

Description

Use this property to specify the style of the X axis. The values for this property are the style classes contained in the style sheet associated with the chart.

Note. Pie chart labels take their style from this property.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

XAxisTicks

Description

Use this property to specify the number of minor tick marks along the X axis. This property takes a numeric value. To control the major tick marks use the XAxisScaleResolution method.

This property is read-write.

See Also

Chapter 11, "Charting Classes," XAxisScaleResolution, page 469

Chapter 11, "Charting Classes," Tick Mark Considerations, page 584

XAxisTitle

Description

Use this property to specify the text for the X-axis title.

This property is read-write.

XAxisTitleOrient

Description

Use this property to specify the orientation of the text for the X-axis title.

You can specify either a number or a constant for this property. The values are:

Numeric Value	*Constant Value*	*Description*
0	%ChartText_Horizontal	Title text is displayed horizontally.
90	%ChartText_Vertical	Title text is displayed vertically. This property is read-write.

XAxisTitleStyle

Description

Use this property to specify the style of the X-axis title. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

XRotationAngle

Description

Use this property to specify the angle of rotation about the X axis in degrees.

This property is valid only with 3D-style charts, that is, charts that have the Type property set as %ChartType_3DBar, %ChartType_3DStackedBar, %ChartType_3DPercentBar, or %ChartType_3DPie.

You do not need to refresh after setting this property. When a method or property is used, the chart is automatically refreshed.

This property takes a string value from "0" to "360".

The chart is rotated in a clockwise direction.

If a negative value is specified the chart is rotated counter-clockwise.

This property is read-write.

YAxisCrossPoint

Description

Use the YAxisCrossPoint property to specify the point at which the Y axis intersects a numeric X axis. The value is in the coordinate system of the X-axis scale.

This property is only applicable to scatter, line or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

This property is read-write.

See Also

Chapter 11, "Charting Classes," XAxisCrossPoint, page 467

YAxisLabelOrient

Description

Use this property to specify whether the Y-axis label is displayed horizontally or vertically.

The default value is horizontal.

You can specify either a numeric value or a constant for this property. Valid values are:

Numeric Value	***Constant Value***	***Description***
0	%ChartText_Horizontal	The Y-axis label is displayed horizontally.
90	%ChartText_Vertical	The Y-axis label is displayed vertically

This property is read-write.

YAxisMax

Description

Use this property to specify the maximum value of the Y (data) axis. This property takes a numeric value.

The maximum value sets the largest value that is displayed on the axis.

Note. If the YAxisMax and YAxisMin properties are not set, or are set to too narrow a range, ticks may not appear on the Y axis.

This property is only applicable to scatter, line, or histogram charts.

This property is read-write.

YAxisMin

Description

Use this property to specify the minimum value of the Y axis. This property takes a numeric value.

The minimum value sets the point at which the chart plots in positive and negative directions.

The default minimum value is the default value of the Y axis.

Note. If the YAxisMax and YAxisMin properties are not set, or are set to too narrow a range, ticks may not appear on the Y axis.

This property is only applicable to scatter, line, or histogram charts.

This property is read-write.

YAxisPrecision

Description

Use the YAxisPrecision property to control the number of decimal places displayed in the labels of a numeric Y axis.

This property is only applicable to scatter, line or histogram charts.

This property is read-write.

See Also

Chapter 11, "Charting Classes," XAxisPrecision, page 469

YAxisScaleResolution

Description

Use the YAxisScaleResolution property to control the level of detail displayed on a numeric axis. Use this property to increase or reduce the number of major ticks and labels on the Y axis.

This property is only applicable to scatter, line, or histogram charts.

The values are:

Numeric Value	***Constant Value***	***Description***
0	%ChartAxisResolution_Fine	Specify fine resolution for the chart, that is, increase the number of major ticks and labels on the Y axis. The fine scale resolution uses extra tick marks to give a more detailed division of the axis. This means that there may not be a one-to-one relationship between the axis labels and the tick marks.
1	%ChartAxisResolution_Coarse	Specify coarse resolution for the chart, that is, reduce the number of major ticks and labels on the Y axis. The coarse scale value maintains a one-to-one relationship between the axis labels and the tick marks.

This property is read-write.

See Also

Chapter 11, "Charting Classes," XAxisScaleResolution, page 469

YAxisStyle

Description

Use this property to specify the style of the Y axis. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

YAxisTicks

Description

Use this property to specify the number of minor tick marks along the Y (data) axis. This property takes a numeric value. To control the major tick marks use the YAxisScaleResolution method.

This property is read-write.

See Also

Chapter 11, "Charting Classes," YAxisScaleResolution, page 474

Chapter 11, "Charting Classes," Tick Mark Considerations, page 584

YAxisTitle

Description

Use this property to specify text of the title for the Y axis.

This property is read-write.

YAxisTitleOrient

Description

Use this property to specify the orientation of the text for the Y-axis title.

You can specify either a number or a constant for this property. The values are:

Numeric Value	*Constant Value*	*Description*
0	%ChartText_Horizontal	Title text is displayed horizontally.
90	%ChartText_Vertical	Title text is displayed vertically. This property is read-write.

YAxisTitleStyle

Description

Use this property to specify the style of the Y-axis title. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

YRotationAngle

Description

Use this property to specify the angle of rotation about the Y axis in degrees.

This property is valid only with 3D-style charts, that is, charts that have the Type property set as %ChartType_3DBar, %ChartType_3DStackedBar, %ChartType_3DPercentBar, or %ChartType_3DPie.

You do not need to refresh after setting this property. When a method or property is used, the chart is automatically refreshed.

This property takes a string value from "0" to "360".

This property is read-write.

ZRotationAngle

Description

Use this property to specify the angle of rotation about the Z axis in degrees.

This property is valid only with 3D-style charts, that is, charts that have the Type property set as %ChartType_3DBar, %ChartType_3DStackedBar, %ChartType_3DPercentBar, or %ChartType_3DPie.

You do not need to refresh after setting this property. When a method or property is used, the chart is automatically refreshed.

This property takes a string value from "0" to "360".

This property is read-write.

Gantt Class Methods

These methods are used by the Gantt class. The methods are described in alphabetic order.

Refresh

Syntax

```
Refresh()
```

Description

Use the Refresh method if the underlying data of the chart has changed, and you want to update the chart.

Note. This method has been deprecated and remains for backward compatibility only. Use the SetTaskAppData method or SetTaskData method instead.

You don't need to refresh after setting a property or method of the chart itself, such as setting the starting point or changing colors. When a method or property is used, the chart is automatically refreshed.

Parameters

None.

Returns

None.

Example

```
&MyChart.Refresh();
```

See Also

Chapter 11, "Charting Classes," SetTaskAppData, page 489 and Chapter 11, "Charting Classes," SetTaskData, page 492

Reset

Syntax

`Reset``()`

Description

Use the Reset method to clear all existing chart settings and data.

Parameters

None.

Returns

None.

See Also

Chapter 11, "Charting Classes," Refresh, page 432

SetActualEndDate

Syntax

`SetActualEndDate``(`*`RecordName.FieldName`*`)`

Description

Use the SetActualEndDate method to specify the field in the record that defines the actual end date. This is an optional method, and is used in conjunction with the SetPlannedEndDate method. If an actual end date is specified, then an actual start date must also be specified. When defined, the actual or baseline tasks are visually shown as a separate task bar and are layered beneath the planned task if the two overlap.

Note. The Gantt chart logs a warning to the file PSJChart.log for any task that has a bad date format in this method or the SetActualStartDate method, but it continues with its normal processing. For these tasks, the overlayed baseline information is not displayed in the chart section.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field and its associated record that contains the value used to define the actual end date. This field must be of DateTime or Date type.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetActualStartDate, page 479; Chapter 11, "Charting Classes," SetPlannedEndDate, page 486 and Chapter 11, "Charting Classes," SetPlannedStartDate, page 486

SetActualStartDate

Syntax

`SetActualStartDate`*`(RecordName.FieldName)`*

Description

Use the SetActualStartDate method to specify the field in the record that defines the actual start date. This is an optional method, and is used in conjunction with the SetPlannedStartDate method. If an actual start date is specified, then an actual end date must also be specified. When defined, the actual or baseline tasks are visually shown as a separate task bar and are layered beneath the planned task if the two overlap.

Note. The Gantt chart logs a warning to the file PSJChart.log for any task that has a bad date format in this method and the SetActualEndDate method, but continues with its normal processing. For these tasks, the overlayed baseline information is displayed in the chart section.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field and its associated record that contains the value used to define the actual start date. This field must be of DateTime or Date type.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetActualEndDate, page 478; Chapter 11, "Charting Classes," SetPlannedEndDate, page 486 and Chapter 11, "Charting Classes," SetPlannedStartDate, page 486

SetActualTaskBarColor

Syntax

SetActualTaskBarColor*(RecordName.FieldName)*

Description

Use the SetActualTaskBarColor method to specify the field in the record that defines the colors for either the actual task bar or the actual milestone glyph.

Parameters

Parameter	*Description*
RecordName.FieldName	Specify the field, and its associated record, that defines the task bar color. This field must be of type number, which means you can only use the numeric value for the color.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetPlannedTaskBarColor, page 487 and Chapter 11, "Charting Classes," TaskMilestoneGlyph, page 517

SetChartArea

Syntax

SetChartArea*(Percentage)*

Description

Use the SetChartArea method to specify what percentage of the Gantt chart is allocated to displaying the chart section. By default, both the task and chart sections are allocated 50% of the available space provided by the Gantt chart.

Parameters

Parameter	*Description*
Percentage	Specify the percentage of the page space for a Gantt chart that is allocated for displaying the chart area. Oracle recommends setting this value above .20 or 20%. Any point below .20 may cause the task bars and task dependency lines to become compressed and hard to read. If the percentage is set less than or equal to .10 or 10%, this parameter automatically takes a value of 0, enabling the table area to occupy the entire Gantt chart area. This parameter takes a float value.

Returns

None.

SetDayFormat

Syntax

`SetDayFormat``(`*`Format`*`)`

Description

Use the SetDayFormat method to specify the format of the day on a DateTime axis.

Parameters

Parameter	*Description*
Format	Specify the format the day should display in. The values are:

Value	*Description*
%Chart_DayFormat	Display the day in a one- or two-digit format.
%Chart_DayFormat_2Digit	Display the day in a two-digit format.

Value	Description
%Chart_DayFormat_Name	Display the day using the full name, such as Monday, Tuesday, and so on.
%Chart_DayFormat_DOY	Display the day of the year in a numeric format, that is 1 — 366.
%Chart_DayFormat_DOY_ 2Digit	Display the day of the year in a two-digit numeric format, that is 01–366.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetHourFormat, page 482; Chapter 11, "Charting Classes," SetMinuteFormat, page 484; Chapter 11, "Charting Classes," SetMonthFormat, page 485 and Chapter 11, "Charting Classes," SetSecondFormat, page 488

SetHourFormat

Syntax

`SetHourFormat`*`(Format)`*

Description

Use the SetHourFormat method to specify how the hour displays on a DateTime axis.

Parameters

Parameter	Description
Format	Specify the format the hours should be displayed in. The values are:

Value	Description
%Chart_HourFormat_24	Display the hour using one or two digits in a 24 hour clock, that is 0–24.
%Chart_HourFormat_24_ 2Digit	Display the hour using two digits for a 24 hour clock, that is 00–24.
%Chart_HourFormat_12	Display the hour using one or two digits using a 12-hour clock. A.M. or P.M. appears next to the value.

Value	*Description*
%Chart_HourFormat_12_ 2Digit	Display the hour using one or two digits using a 12-hour clock. A.M. or P.M. appears next to the value.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDayFormat, page 481; Chapter 11, "Charting Classes," SetMinuteFormat, page 484 and Chapter 11, "Charting Classes," SetSecondFormat, page 488

SetLegend

Syntax

`SetLegend`*`(&Array_of_String)`*

Description

Use the SetLegend method to specify alternative legends. By default, the legends are populated from the record field specified with SetDataSeries.

This method is used to write both the overlay legend and the series legend. The labels are overwritten in the order of elements in the array, that is, the first element overwrites the first series, the second overwrites the second, and so on, with the last element in the array overwriting the overlay legend. There can be more than one series in an overlay.

If you do not specify an element for an array (that is, a blank or a null) then no legend is listed for that series.

Note. Default label text is not automatically translated. If you set your own labels, be sure to use translated text, such as message catalog entries.

Parameters

Parameter	*Description*
&Array_of_String	Specify an already instantiated array of string, containing the text that you want to use for the legend.

Returns

None.

Example

The following example displays a legend containing only the overlay.

```
&LegendArray = CreateArray("", "", "Revenue");
```

```
&MyChart.SetLegend(&LegendArray);
```

See Also

Chapter 11, "Charting Classes," SetDataSeries, page 443

Chapter 8, "Array Class," page 257

SetMinuteFormat

Syntax

SetMinuteFormat(*Format*)

Description

Use the SetMinuteFormat method to specify how minutes are displayed on the DateTime axis.

Locale-specific time separators are used to format the time.

Parameters

Parameter	***Description***
Format	Specify how the minutes should be displayed:

Value	***Description***
%Chart_MinuteFormat	Displays the minutes using a one- or two-digit number.
%Chart_MinuteFormat_2Digit	Displays the minutes using a two-digit number.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetHourFormat, page 482 and Chapter 11, "Charting Classes," SetSecondFormat, page 488

SetMonthFormat

Syntax

`SetMonthFormat``(`*`Format`*`)`

Description

Use the SetMonthFormat method to specify the format of months on a DateTime axis.

Parameters

Parameter	*Description*
Format	Specify the format of how the month should be displayed. Values are:

Value	*Description*
%Chart_MonthFormat	Display the month using a one- or two-digit number.
%Chart_MonthFormat_2Digit	Display the month using a two-digit number.
%Chart_MonthFormat_FullName	Display the month using the full name.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDayFormat, page 481 and Chapter 11, "Charting Classes," SetYearFormat, page 505

SetPlannedEndDate

Syntax

`SetPlannedEndDate(RecordName.FieldName)`

Description

Use the SetPlannedEndDate method to specify the field in the record that defines the planned end date. If planned versus actual dates are not used, this method specifies the end date of the task.

This method is required when creating a Gantt chart.

Note. The Gantt chart logs an error to the file PSJChart.log and stops its normal processing if it finds a bad date format in any of the rows in the field associated with this method and the SetPlannedStartDate method.

Parameters

Parameter	***Description***
RecordName.FieldName	Specify the field and its associated record that contains the value used to define the planned end date. This field must be of DateTime or Date type.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetActualEndDate, page 478; Chapter 11, "Charting Classes," SetActualStartDate, page 479 and Chapter 11, "Charting Classes," SetPlannedStartDate, page 486

SetPlannedStartDate

Syntax

`SetPlannedStartDate(RecordName.FieldName)`

Description

Use the SetPlannedStartDate method to specify the field in the record that defines the planned start date. If planned versus actual dates are not used, this method specifies the start date of the task.

This method is required when creating a Gantt chart.

Note. The Gantt chart logs an error to the file PSJChart.log and stops its normal processing if it finds a bad date format in any of the rows in the field associated with this method and the SetPlannedEndDate method.

Parameters

Parameter	***Description***
RecordName.FieldName	Specify the field and its associated record that contains the value used to define the planned start date. This field must be of DateTime or Date type.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetActualEndDate, page 478; Chapter 11, "Charting Classes," SetPlannedEndDate, page 486 and Chapter 11, "Charting Classes," SetPlannedStartDate, page 486

SetPlannedTaskBarColor

Syntax

`SetPlannedTaskBarColor`*`(RecordName.FieldName)`*

Description

Use the SetPlannedTaskBarColor method to specify the field in the record that defines the colors for either the planned task bar or the planned milestone glyph.

Parameters

Parameter	***Description***
RecordName.FieldName	Specify the field, and its associated record, that defines the task bar color. This field must be of type number, which mean you can only use the numeric values for specifying color.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetActualTaskBarColor, page 480 and Chapter 11, "Charting Classes," TaskMilestoneGlyph, page 517

SetSecondFormat

Syntax

`SetSecondFormat``(Format)`

Description

Use the SetSecondFormat method to specify how seconds display on a DateTime axis.

Locale-specific time separators are used to format the time.

Parameters

Parameter	***Description***
Format	Specify the format for how the seconds should be displayed:

Value	***Description***
%Chart_SecondFormat	Displays the seconds using a one- or two-digit number.
%Chart_SecondFormat_2Digit	Displays seconds using a two-digit number.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetHourFormat, page 482 and Chapter 11, "Charting Classes," SetMinuteFormat, page 484

SetTableXScrollbar

Syntax

SetTableXScrollbar*(Scroll_ArrowAmount)*

Description

Use the SetTableXScrollbar method to specify the amount that the scroll bar in the table section of the Gantt chart scroll data when the user clicks on the arrows or scrollbar.

Parameters

Parameter	***Description***
Scroll_ArrowAmount	Specify the amount to scroll in the table section of the Gantt chart when the user clicks on the horizontal scrollbar arrow. The scroll amount must be a value between zero and one. The default value is 0.10. The amount specified indicates the percentage of the table section that is scrolled when the user clicks the scroll bar.

Returns

None.

SetTaskAppData

Syntax

SetTaskAppData*(RecordName.FieldName1* [, *RecordName.FieldName2* [,*RecordName.FieldName3. . .*]])

Description

Use the SetTaskAppData method to specify the fields in the record that define the application data to be viewed in the table section.

In the table section, column one (or the left-most column) always displays the work breakdown structure (WBS) numbering (if given) and name of the task (or task ID if the name is not given). This method allows applications to define additional task data columns to be displayed in the right-most columns of the table section.

Parameters

Parameter	*Description*
RecordName.FieldNames	Specify the fields and their associated record that defines one or more application specific fields to be displayed as a column in the table section. The order in which the fields are given as parameters defines their column order in the table section. Because the task name (or task ID if the name is not given) is always shown in column one, the first record field value is displayed in column two.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskAppDataTitles, page 490; Chapter 11, "Charting Classes," SetTaskData, page 492 and Chapter 11, "Charting Classes," SetTaskDependencyData, page 493

SetTaskAppDataTitles

Syntax

SetTaskAppDataTitles*(&TitleArray)*

Description

Use the SetTaskAppDataTitles method to specify the column titles to be displayed in the table section, starting with column two. The title for column one (that is, the left-most column) in the table section is set using the SetTaskTitle method.

The length of the array should match the number of application data fields specified with the SetTaskAppData method. Additional strings beyond the number of application data fields are not displayed. If the number of strings in the array specified by this method is less than the number of application data fields, the blanks are displayed in these title columns.

Parameters

Parameter	*Description*
&TitleArray	Specify an array of strings containing the column titles to be displayed in the table section.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskAppData, page 489 and Chapter 11, "Charting Classes," TaskTitle, page 517

SetTaskBarURL

Syntax

`SetTaskBarURL`*`(RecordName.FieldName)`*

Description

Use the SetTaskBarURL method to specify the field in the record which defines the URL the browser redirects to when the user clicks on a task bar.

By default, the FieldChange event on the planned end date field of the Gantt task data record is triggered if the image map based interactivity is turned on (using the IsDrillable property). This can be overridden by specifying a URL that is triggered when the task bar is clicked.

Use the SetTaskDependencyURL method to specify the URL to be used when the user clicks on a dependency line in the chart section.

Parameters

Parameter	*Description*
Record.Fieldname	Specify the field, and its associated record, that defines the URL the browser is redirected to when the task bar is clicked. This field must be of type character. The URL must be an absolute URL.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskDependencyURL, page 496

Chapter 11, "Charting Classes," IsDrillable, page 457

SetTaskData

Syntax

`SetTaskData({Record.RecordName | &Rowset})`

Description

Use the SetTaskData method to specify from where the task data is to be populated, either from a record or an already instantiated rowset.

Note. If you specify a rowset, the data in the rowset must be provided in the correct display order.

This method is required when you're creating a Gantt chart.

Parameters

Parameter	Description
RecordName | *&Rowset*	Specify the record or rowset that contains the task data. If you specify a record, you must prefix the record name with the keyword **Record.**

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskDependencyData, page 493

SetTaskDependencyChildID

Syntax

`SetTaskDependencyChildID(RecordName.FieldName)`

Description

The *dependent* task is the task where the dependency arrow ends. Use the SetTaskDependencyChildID method to specify the field in the record that defines the task ID for the dependent task.

The ID must match up to an ID in the Task data set.

This method is required if you specify dependency data.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the task ID for the dependant task. This field must be of type number.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskDependencyData, page 493; Chapter 11, "Charting Classes," SetTaskDependencyURL, page 496 and Chapter 11, "Charting Classes," SetTaskDependencyParentID, page 494

SetTaskDependencyData

Syntax

```
SetTaskDependencyData({RecordName | &Rowset})
```

Description

Use the SetTaskDependencyData method to specify either a record, or an already instantiated rowset, that contains the dependency data.

Note. If you specify a rowset, the data in the rowset must be provided in the correct display order.

Parameters

Parameter	Description
RecordName | &Rowset	Specify the record or rowset that contains the task data. If you specify a record, you must prefix the record name with the keyword **Record.**

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskData, page 492; Chapter 11, "Charting Classes," SetTaskDependencyChildID, page 492; Chapter 11, "Charting Classes," SetTaskDependencyURL, page 496 and Chapter 11, "Charting Classes," SetTaskDependencyParentID, page 494

SetTaskDependencyParentID

Syntax

`SetTaskDependencyParentID``(RecordName.FieldName)`

Description

The *depender* task is the task where the dependency arrow originates. Use the SetTaskDependencyParentID method to specify the field in the record that defines the task ID for the depender task.

The ID must match up to an ID in the Task data set.

This method is required if you specify dependency data.

Parameters

Parameter	***Description***
RecordName.FieldName	Specify the field and its associated record that defines the task ID for the depender task. This field must be of type number.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskDependencyChildID, page 492 and Chapter 11, "Charting Classes," SetTaskDependencyData, page 493

SetTaskDependencyType

Syntax

`SetTaskDependencyType``(RecordName.FieldName)`

Description

Use the SetTaskDependencyType method to specify the type of dependency relationship between predecessor and successor tasks.

The values for type are as follows:

Type	***Description***
Finish-to-Start	The arrow starts at the end of the predecessor task bar and ends at the start of the successor task bar. This is the default value
Finish-to-Finish	The arrow starts at the end of the predecessor task bar and ends at the end of the successor task bar.
Start-to-Start	The arrow starts at the start of the predecessor task bar and ends at the start of the successor task bar.
Start-to-Finish	The arrow starts at the start of the predecessor task bar and ends at the end of the successor task bar.

Parameters

Parameter	***Description***
RecordName.FieldName	Specify the field, and its associated record, that defines the type of dependencies between predecessor and successor tasks. This field must be of type integer, and one of the following values:

Value	***Description***
0 or any value greater than 3	Finish-to-Start
1	Finish-to-Finish
2	Start-to-Start
3	Start-to-Finish

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskDependencyChildID, page 492; Chapter 11, "Charting Classes," SetTaskDependencyData, page 493 and Chapter 11, "Charting Classes," SetTaskDependencyParentID, page 494

SetTaskDependencyURL

Syntax

`SetTaskDependencyURL``(`*`RecordName.FieldName`*`)`

Description

Use the SetTaskDependencyURL method to specify the field in the record that defines the URL the browser redirects to when the user clicks on a task bar.

By default, the FieldChange event on the child ID field of the task dependency record is triggered if the image map based interactivity is turned on (using the IsDrillable property). This can be overridden by specifying a URL that is triggered when the task bar is clicked.

Use the SetTaskBarURL method to specify the URL to be used when the user clicks on a task bar in the chart section.

Parameters

Parameter	***Description***
RecordName.FieldName	Specify the field, and its associated record, that defines the URL the browser is redirected to when the dependency arrow is clicked. This field must be of type character. The URL must be an absolute URL.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskBarURL, page 491; Chapter 11, "Charting Classes," SetTaskDependencyChildID, page 492; Chapter 11, "Charting Classes," SetTaskDependencyData, page 493 and Chapter 11, "Charting Classes," SetTaskDependencyParentID, page 494

SetTaskDrill

Syntax

`SetTaskDrill``(Recordname.FieldName)`

Description

Use the SetTaskDrill method to specify for which field PeopleCode FieldEdit and FieldChange events will execute when the user clicks the task bar.

The field can be any field that is loaded in the component buffer.

The system does not change the value in the field when the user clicks the field. Any changes or other processing for the field must be done within the PeopleCode program.

Parameters

Parameter	*Description*
RecordName.FieldName	Specify the record name and the field name of the field to be used for field change processing. The field must be available in the component buffer.

Returns

None.

SetTaskExpanded

Syntax

`SetTaskExpanded``((RecordName.FieldName)`

Description

Use the SetTaskExpanded method to specify the field in the record that defines the expanded state of parent tasks.

Whenever an end user changes the expanded state by clicking on the expand (collapse) icon in the table section, the expanded state in the rowset is updated. In addition, a field change event occurs to allow applications to synchronize expanded state changes with other parts of their application.

Parameters

Parameter	*Description*
RecordName.FieldName	Specify the field, and its associated record, that defines the text in the task hint pop-up. This field must be of type character, that is, Y or N.

Returns

None.

SetTaskHints

Syntax

`SetTaskHints``(`*`RecordName.FieldName`*`)`

Description

Use the SetTaskHints method to specify the field in the record which defines the task hint pop-up text. The description is displayed in a pop-up bubble when a cursor hovers over the task bar or milestone glyph.

Parameters

Parameter	*Description*
RecordName.FieldName	Specify the field, and its associated record, that defines the text in the task hint pop-up. This field must be of type character.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskID, page 499 and Chapter 11, "Charting Classes," SetTaskName, page 502

SetTaskID

Syntax

`SetTaskID``(`*`Recordname.FieldName`*`)`

Description

Use the SetTaskID method to specify the field that defines the unique task identifier. The task ID is used to support task linking and dependencies.

This method is required when you're creating a Gantt chart.

Parameters

Parameter	***Description***
RecordName.FieldName	Specify the record name and the field name of the field that defines the unique task identifier. The field must be of type number.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskName, page 502

SetTaskLabel

Syntax

`SetTaskLabel``(`*`RecordName.FieldName`*`)`

Description

Use the SetTaskLabel method to specify the field in the record that defines the task label. The label is displayed alongside its corresponding task bar in the chart area. If a task label is not provided, the task name is used instead. If a task name is not provided, the task id is used.

Parameters

Parameter	*Description*
RecordName.FieldName	Specify the field, and its associated record, that contains the task label. This field must be of type character.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskID, page 499 and Chapter 11, "Charting Classes," SetTaskName, page 502

SetTaskLevel

Syntax

SetTaskLevel*(RecordName.FieldName)*

Description

Use the SetTaskLevel method to specify the field in the record that defines the level for the task. The outermost summary tasks are always defined as task level one. Tasks with task levels greater than one are subtasks. Subtasks may also contain other subtasks.

If a field is not provided to define the level, all tasks are defined at level 1. A maximum of 32 levels are supported.

The following table shows an example of the different levels that could be used, as well as what the parent task level is.

Note. The parent task is not actually part of the data.

Task Name	*Task Level*	*Parent Level*
Phase I	1	None
Evaluation	2	Phase I
Research	3	Evaluation
Report Findings	3	Evaluation

Task Name	*Task Level*	*Parent Level*
Create Budget	2	Phase I
Approve Budget	3	Create Budget
Phase II	1	None
Code and Test	2	Phase II
Phase III	1	None
Training	2	Phase III

Parameters

Parameter	*Description*
RecordName.FieldName	Specify the field, and its associated record, that contains the information about the task level. This field must be of type number. Up to 32 levels are supported.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskID, page 499

SetTaskMilestone

Syntax

`SetTaskMilestone`*`(RecordName.FieldName)`*

Description

Use the SetTaskMilestone method to specify the field in the record that defines whether the task should be treated as a milestone.

The fields specified by the SetPlannedStartDate and SetActualStartDate methods are used for determining the dates for the planned and actual milestones. The other date fields in the rowset are ignored

Parameters

Parameter	*Description*
RecordName.FieldName	Specify the field and its associated record that defines whether the task should be treated as a milestone. The field must be of type Character, and be one character long. The possible values for this field are Y, used to specify that this task is a milestone, or N, if not.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetActualStartDate, page 479; Chapter 11, "Charting Classes," SetTaskMilestone, page 501 and Chapter 11, "Charting Classes," SetPlannedStartDate, page 486

SetTaskName

Syntax

`SetTaskName``(RecordName.FieldName)`

Description

Use the SetTaskName method to specify the field in the record that defines the task name. If you do not specify a task name, the task ID is used in the table section of the Gantt chart.

Although this method is not required, Oracle recommends using it to display meaningful information in the table section of the Gantt chart.

Parameters

Parameter	*Description*
RecordName.FieldName	Specify the field name and its associated record that defines the task name. The field must be of type character.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskID, page 499

SetTaskProgress

Syntax

`SetTaskProgress`*`(RecordName.FieldName)`*

Description

Use the SetTaskProgress method to specify the field in the record that defines the length of the progress bar for this task as shown in the chart area of the Gantt chart.

The value in the field must be between 0 and 100, inclusive. Zero indicates no progress bar is shown, 100 indicates that the progress bar should cover the entire length of the task bar. Values greater than 100 are automatically converted to 100.

Parameters

Parameter	*Description*
RecordName.FieldName	Specify the field, and its associated record, that defines the length of the progress bar for this task. This field must be of type number. The value in the field must be between 0 and 100, inclusive. You must not specify a value greater than 100.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetTaskProgressBarColor, page 503

SetTaskProgressBarColor

Syntax

`SetTaskProgressBarColor`*`(RecordName.FieldName)`*

Description

Use the SetTaskProgressBarColor method to specify the field in the record that defines the color for the progress bar.

For the progress bar to be visible, its color must contrast with the task bar color. No borders are drawn around the progress bar.

Parameters

Parameter	***Description***
RecordName.FieldName	Specify the field, and its associated record, that defines the task progress bar color. This field must be of type numeric, which means that you can only use the numeric value for the chart colors, not the constant.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetActualTaskBarColor, page 480 and Chapter 11, "Charting Classes," SetTaskProgress, page 503

SetWBSNumbering

Syntax

`SetWBSNumbering``(`*`RecordName.FieldName`*`)`

Description

Use the SetWBSNumbering method to specify the field in the record that defines the WBS numbering to be displayed for each task row in the table section of the Gantt chart.

A WBS is very similar in structure and layout to a document outline. Each item at a specific level of a WBS is numbered consecutively (that is, 10, 20, 30, 40, 50). Each item at the next level is numbered within the number of its parent item (that is, 10.1, 10.2, 10.3, 10.4). For example:


```
1.
   1.1
      1.1.1
      1.1.2
      1.1.3
   1.2
      1.2.1
      1.2.2
2.
. . .
```

If no field name is provided, WBS numbering is not displayed along with the tasks.

Regardless if WBS numbering is provided, child tasks are indented to the right of their parent task.

Parameters

Parameter	***Description***
RecordName.FieldName	Specify the field name and its associated record that defines the WBS numbering to be displayed. The field must be of type character.

Returns

None.

SetYearFormat

Syntax

`SetYearFormat``(`*`Format`*`)`

Description

Use the SetYearFormat method to specify the format of years on the DateTime axis.

Parameters

Parameter	***Description***
Format	Specify the format of how the year should be displayed. Values are:

Value	***Description***
%Chart_YearFormat	Display the year in a numeric one- or two-digit format.

Value	*Description*
%Chart_YearFormat_2Digit	Display the year in a numeric two-digit format.
%Chart_YearFormat_4Digit	Display the year in a numeric four-digit format.

Returns

None.

See Also

Chapter 11, "Charting Classes," SetDayFormat, page 481 and Chapter 11, "Charting Classes," SetMonthFormat, page 485

Gantt Class Properties

These properties are used by the Gantt class. The properties are described in alphabetic order.

AxisEndDateTime

Description

Use this property to specify the date and time where the chart axis should end. The value must be of type Date or DateTime.

This property is read-write.

AxisStartDateTime

Description

Use this property to specify the date and time where the chart axis should begin. The value must be of type Date or DateTime.

This property is read-write.

DataEndDateTime

Description

Use this property to specify where the chart should stop displaying data. The value must be of type DateTime.

This property is read-write.

DataStartDateTime

Description

Use this property to specify where the chart should start displaying data. The value must be of type DateTime.

This property is read-write.

DataStartRow

Description

This property sets or returns the row number (after any preprocessing) at which to start plotting. This is useful when you have many rows of data in a rowset, and you want to start at a particular row. This can also be useful for creating your own 'scrolling' effect, by specifying both the start row and how many rows to be displayed (by using the DataWidth property).

If this property is not set, then the value is 1.

This property is read-write.

See Also

Chapter 11, "Charting Classes," DataWidth, page 454

DataWidth

Description

This property returns or sets the number of rows to plot for a series. The starting point for the number of rows is set with the DataStartRow property. If the DataWidth property value is not set, then the width is from the DataStartRow to the last element in the rowset. If the property value is less than zero, then it will automatically be set to one.

This property is read-write.

Example

Suppose your rowset returned 400 rows. You wouldn't want all 400 to display in your chart. Instead, you'd pick a subset of those rows. The row to start with is set with DataStartRow, while how many rows to display is set with DataWidth.

The following code could be used with a push button connected with a chart. The push button enables the next set of data to be displayed with the chart. You don't need to refresh after setting this property. When a method or property is used, the chart is automatically refreshed.

```
Local Chart &MyChart;

&MyChart = GetChart(ChartRec.DisplayField);
&Start = &MyChart.DataStartRow;
/* display the next 20 rows of data */
&MyChart.DataStartRow = &Start + 20;
&MyChart.DataWidth = 20;
```

See Also

Chapter 11, "Charting Classes," DataStartRow, page 454

GridLines

Description

Use this property to specify whether to show grid lines with the chart.

Note. This property is ignored unless you have set RevertToPre850 to True for the chart.

Gantt charts only support the %ChartGrid_Vertical and %ChartGrid_None values.

Grid lines are shown within the chart section of the Gantt chart, if enabled, and are tied to the major tick mark. The major tick mark is associated with the smallest time increment displayed on the DateTime axis using the SetFormat functions such as the SetMonthFormat, SetDayFormat, and so on. For example, if you choose to display both years and months on the DateTime axis, the major tick marks denote the months.

If you do not specify a value, no grid lines are shown.

This property controls only the display of grid lines. If you want to specify the style of the grid lines, use the GridLineType property.

The values are:

Numeric Value	***Constant Value***	***Description***
0	%ChartGrid_None	Don't show grid lines (default)
2	%ChartGrid_Vertical	Show vertical grid lines

This property is read-write.

See Also

Chapter 11, "Charting Classes," GridLineType, page 455

GridLineType

Description

Use this property to specify the style of the grid lines when either or both vertical or horizontal grid lines are turned on.

Note. This property is ignored unless you have set RevertToPre850 to True for the chart.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

The default value is %ChartLine_Solid. The possible values are:

Value	***Description***
%ChartLine_Solid	Draw grid lines with solid lines.
%ChartLine_Dash	Draw grid lines with lines composed of dashes.
%ChartLine_Dot	Draw grid lines with lines composed of dots.
%ChartLine_DashedDot	Draw grid lines with lines composed of both dashes and dots.

This property is read-write.

HasLegend

Description

Use this property to specify if a legend is displayed with the chart. This property takes a Boolean value: True, if the legend is displayed with the chart, False otherwise.

The default value is False.

This property is read-write.

Height

Description

Use this property to specify the height of the chart. This property takes a numeric value. The unit of measurement is pixels.

Generally this property is used with charts created for iScripts, to specify the height of the image, before generating the image map. You can also use it to overwrite the height set in PeopleSoft Application Designer.

If you try to read this property before setting it, the value returned is zero.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Height, page 456

ImageMap

Description

This property returns the HTML client-side image map. The image map is what makes the chart interactive.

This property is read-only.

See Also

Chapter 11, "Charting Classes," Creating a Chart Using an iScript, page 596

InteractiveEnd

Description

Use this property to specify whether the user can change the end date of a task by dragging and dropping.

This property takes a Boolean value: True if the user can change the end date, False otherwise.

The default value of this property is False.

This property is read-write.

InteractiveMove

Description

Use this property to specify whether the user can move a task bar by dragging and dropping.

This property takes a Boolean value: True if the user can move the bar, False otherwise.

The default value of this property is True.

This property is read-write.

InteractiveProgress

Description

Use this property to specify whether the user can change the progress value of a task by dragging and dropping.

This property takes a Boolean value: True if the user can change the progress value, False otherwise.

The default value of this property is True.

This property is read-write.

InteractiveStart

Description

Use this property to specify whether the user can change the start date of a task by dragging and dropping.

This property takes a Boolean value: True if the user can change the start date, False otherwise.

The default value of this property is False.

This property is read-write.

IsDrillable

Description

The IsDrillable property is used with the task bars as well as the task dependency lines in the chart section of a Gantt chart. Where the user is directed depends on whether a URL is provided or not, as detailed in this table:

User Action	*If URL is not provided:*	*If URL is provided:*
Clicks a task bar.	FieldChange event is associated with the Planned End Date field in the Task table.	Redirects to URL provided by field identified using the SetTaskBarURL method.
Clicks a task dependency line.	FieldChange event is associated with the Child ID field in the Task Dependency table.	Redirects to URL provided by field identified using the SetTaskDependencyURL method.
Clicks an expand/collapse icon.	FieldChange event is associated with the task data.	Redirects to URL provided by field identified using the SetTaskBarURL method.

This property takes a Boolean value: True if the end-user can click the chart to initiate an action, and False otherwise.

Note. Setting this property has no effect if the IsPlainImage property is set as True.

Note. When using a chart in an iScript, this property must be used in conjunction with the SetDataURLs method. The SetDataURLs method must point to a field that is populated with the URLs to be triggered for each data point in the chart.

The default value for this property is False.

This property is read-write.

See Also

Chapter 11, "Charting Classes," IsPlainImage, page 512

IsPlainImage

Description

Use this property to specify whether the chart is built without any extra HTML, such as SRC or MAP tags.

If this property is specified as True, the end-user will not be able to click on the chart to trigger an event. In addition, the pop-up data hints will not appear when the end-user passes a mouse over the chart. However, you may see a performance improvement if your chart has an extremely complicated image map and you specify this property as True.

This property takes a Boolean value: True if the chart is built without extra HTML, False otherwise.

The default value of this property is False.

This property is read-write.

LegendPosition

Description

Use this property to specify where the legend should appear, in relationship to the chart. You can specify either a numeric or constant value for this property.

Note. ChartLegend_Separate generates a legend without a chart. The legend takes over the entire charting area. The legend appears in the center of the chart.

The values are:

Numeric Value	*Constant Value*	*Description*
0	%ChartLegend_Left	Display legend to the left of the chart.
1	%ChartLegend_Right	Display legend to the right of the chart.
2	%ChartLegend_Top	Display legend on top of the chart.
3	%ChartLegend_Bottom	Display legend below the chart.
4	%ChartLegend_Separate	Generate a legend without a chart. This property is read-write.

LegendStyle

Description

Use this property to specify the style of the legend. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

MainTitle

Description

Use this property to specify the text for the main title of the chart.

This property takes a string value.

This property is read-write.

MainTitleStyle

Description

Use this property to specify the style of the main title. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

PixelsPerRow

Description

This property returns an integer value of the row height (in pixels) for each row in the table section. Use this property to properly size the height of the table section of your Gantt chart so that all rows in the table are displayed. You can do so by multiplying the value for the PixelsPerRow property by the number of visible rows in the rowset.

The system calculates the value of the PixelsPerRow property at runtime after the Gantt image is rendered. To do so, this property must be read after the PeopleCode Activate event. This can be done by reading in the PixelPerRow value during a FieldChange event on a refresh button on the page.

This property is read-only.

RevertToPre850

Description

PeopleTools Version 8.50 implements new chart style classes that bring a consistent look and feel to charts.

Use this property to specify whether the chart will revert to the pre-PeopleTools 8.50 chart style classes.

Warning! By default RevertToPre850 is set to False, which means that charts that were created prior to PeopleTools 8.50 will use the new 8.50 chart style classes.

Any custom Chart class properties that were set in PeopleCode for existing charts that use the new 8.50 chart style classes will be ignored if those properties are defined in the 8.50 chart style classes.

Oracle strongly recommends that you review existing charts to verify that they display correctly.

This property takes a Boolean value:

Parameter	***Description***
True	Do not use the PeopleTools 8.50 style classes for this chart so that the chart will maintain its original appearance.
False	Use the PeopleTools 8.50 style classes for this chart. Any chart properties that are set in application PeopleCode may be overridden by the 8.50 style classes.

The default is False.

This property is read-write.

See Also

Chapter 11, "Charting Classes," OLLineType, page 462

ShowTaskLabels

Description

Use this property to specify whether or not task labels are displayed alongside the corresponding task bar in the chart area. This property takes a boolean value; true to display the labels, false to not display the labels. The default is to display the labels.

This property is read-write.

Style

Description

A *style sheet* is a definition, like a record or field definition, that you create in PeopleSoft Application Designer. It contains a collection of formatting styles, called *classes,* each of which can be applied to either the entire chart or to specific elements within a chart.

Use this property to specify the overall style of the chart by specifying a style class. The value must be a valid class within the style sheet specified for the chart.

Individual style properties (such as SetXAxisStyle, MainTitleStyle, and so on) can be used to override this value for individual elements of the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

StyleSheet

Description

Use this property to associate a style sheet with a chart.

The PTSTYLEDEF style sheet is the default style sheet associated with a chart.

Both the Style property for the chart, and individual style properties (such as SetXAxisStyle, MainTitleStyle, and so on) can be used to override this value for individual elements of the chart.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

TaskDependencyLineType

Description

Use this property to specify an integer value that specifies the line type of the line connecting dependent tasks. This value affects all the dependency lines for the entire Gantt chart, not just for a specific set of tasks.

This property is read-write.

See Chapter 11, "Charting Classes," Using Style Sheets with PeopleTools Version 8.50 Charts, page 386 and Chapter 11, "Charting Classes," RevertToPre850, page 463.

Numeric Value	***Constant Value***	***Description***
0	%ChartLine_Solid	Solid line
1	%ChartLine_Dashed	Dashed line
2	%ChartLine_Dotted	Dotted line
3	%ChartLine_Mixed	Mixture of dashes and dots

TaskMilestoneGlyph

Description

Use this property of type integer to specify which field in the record defines the glyph to represent a task milestone (for both planned and actual milestones). This glyph is used for the entire chart, not just for a specific task.

By default a diamond shaped glyph will be used.

Milestones associated with planned dates use colors defined by the SetPlannedTaskBarColor method. Milestones associated with actual dates use colors defined by the SetActualTaskBarColor method.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Using Gantt Glyphs, page 405

TaskTitle

Description

Use this property to specify the column title to be displayed in column one (that is, the left-most column) in the table section of the Gantt chart. Use the SetTaskAppDataTitles method to specify the column titles for the application specific data fields (that is, column two and beyond).

This property is read-write.

See Also

Chapter 11, "Charting Classes," SetTaskAppDataTitles, page 490

Width

Description

Use this property to specify the width of the chart. This property takes a numeric value. The unit of measurement is pixels.

Generally this property is used with charts created for iScripts, to specify the width of the image, before generating the image map. You can also use it to overwrite the width set in PeopleSoft Application Designer.

If you try to read this property before setting it, the value returned is zero.

This property is read-write.

XAxisPosition

Description

Use this property to specify the position of the X axis relative to the Y axis.

Note. This property is not supported in Gantt charts.

You can use either a constant or a numeric value for this property. Valid values are:

Numeric Value	***Constant Value***	***Description***
1	%ChartAxis_ZeroPosition	Positions the axis in line with the 0 on the opposite axis. Default position for the axis.
2	%ChartAxis_LowPosition	Positions the axis in line with the lowest value on the opposite axis.
3	%ChartAxis_HighPosition	Positions the axis in line with the highest value on the opposite axis.

This property is read-write.

YAxisPosition

Description

Use this property to specify the position of the Y axis relative to the X axis.

Note. This property is not supported in Gantt charts.

You can use either a constant or a numeric value for this property. Valid values are:

Numeric Value	***Constant Value***	***Description***
1	%ChartAxis_ZeroPosition	Positions the axis in line with the 0 on the opposite axis. Default position for the axis.
2	%ChartAxis_LowPosition	Positions the axis in line with the lowest value on the opposite axis.
3	%ChartAxis_HighPosition	Positions the axis in line with the highest value on the opposite axis.

This property is read-write.

OrgChart Class Methods

These methods are used by the OrgChart class. The methods are described in alphabetic order.

SetCrumbData

Syntax

`SetCrumbData``(&Rowset)`

Description

Use the SetCrumbData method to specify the source for the data for the organization chart breadcrumbs. Use an already instantiated and populated level-1 component rowset that contains the breadcrumb data.

Parameters

Parameter	***Description***
&Rowset	A level-1 rowset populated with the breadcrumbs data.

Returns

None.

SetCrumbRecord

Syntax

SetCrumbRecord(Record.*Record_Name*)

Description

Use the SetCrumbRecord method to specify the derived/working record name that contains the information about the breadcrumbs.

The breadcrumb record is an application-specific derived/work record created using a clone of the PTORGCRMB_SBR subrecord definition.

Parameters

Parameter	***Description***
Record.*Record_Name*	Specify the name of the derived/work record that contains the data for the breadcrumbs. You must include the **Record** keyword.

Returns

None.

SetDropdownData

Syntax

SetDropdownData *(&Rowset)*

Description

Use the SetDropdownData method to specify the source for the data for the organization chart drop-down list boxes. Use an already instantiated and populated level-1 component rowset that contains the drop-down list data.

Parameters

Parameter	*Description*
&Rowset	A level-1 rowset populated with the drop-down list data.

Returns

None

SetDropdownRecord

Syntax

SetDropDownRecord**(Record.***RecordName*)

Description

Use the SetDropdownRecord method to specify the derived/working record name that contains the information about the dropdown lists.

The dropdown record is an application-specific derived/work record created using a clone of the PTORGBOXLIST_SBR subrecord definition.

Parameters

Parameter	*Description*
Record_Name	Specify the name of the derived/work record that contains the data for the drop down lists. You must include the **Record** keyword.

Returns

None

SetLegend

Syntax

`SetLegend`*`(&Array_of_String)`*

Description

Use the SetLegend method to specify legend text. Legend text labels the images set using SetLegendImg.

If you do not specify an element for an array (that is, a blank or a null) then no legend is listed for that node.

Note. Legend text is not automatically translated. If you set your own labels, be sure to use translated text, such as message catalog entries.

Parameters

Parameter	*Description*
&Array_of_String	Specify an already instantiated array of string, containing the text that you want to use for the legend.

Returns

None.

Example

```
&LegendArray = CreateArray("A", "B", C", "D ");
&ocOrgChart.SetLegend (&LegendArray);
```

See Also

Chapter 11, "Charting Classes," SetDataSeries, page 443

Chapter 8, "Array Class," page 257

SetLegendImg

Syntax

SetLegendImg*(&Array_of_String)*

Description

Use the SetLegendImg method to specify the legend image names.

Parameters

Parameter	*Description*
&Array_of_String	An already instantiated array containing the names of the images that you want to use with the legend.

Returns

None.

SetNodeData

Syntax

`SetNodeData``(&Rowset)`

Description

Use the SetNodeData method to specify the source for the node data for the organization chart. Use an already instantiated and populated level-1 component rowset that contains the node data.

This is a required method to build an organization chart.

If you make a change to the underlying data of a chart, call the SetNodeData method again to update the chart.

Parameters

Parameter	*Description*
&Rowset	A level-1 component rowset populated with the node data. This rowset references the node record of the chart.

Returns

None.

SetNodeRecord

Syntax

SetNodeRecord(Record.*Record_Name***)**

Description

Use the SetNodeRecord method to specify the organization node record.

The node record is an application-specific derived/work record created using a clone of the PTORGNODE_SBR subrecord definition.

This method is required to build an organization chart.

Parameters

Parameter	***Description***
Record.*Record_Name*	Specify the name of the derived/work record that contains the data for the organization node. You must include the **Record** keyword.

Returns

None.

SetPopUpNodeData

Syntax

SetPopUpNodeData*(&Rowset)*

Description

Use the SetPopUpNodeData method to specify the source for the pop-up node data for the organization chart. Use an already instantiated and populated level-1 component rowset that contains the pop-up node data.

This method is not required if no pop-up chart is available to be displayed in the organization chart.

If you make a change to the underlying data of a pop-up chart, call the SetPopUpNodeData method again to update the chart.

Parameters

Parameter	*Description*
&Rowset	A level-1 rowset populated with the pop-up node data. This rowset references the pop-up node record of the chart.

Returns

None.

SetPopUpNodeRecord

Syntax

SetPopUpNodeRecord(Record.*Record_Name***)**

Description

Use the SetNodeRecord method to specify the pop-up node record.

The node record is an application-specific derived/work record created using a clone of the PTORGPOPUP_SBR subrecord definition.

This method is not required if no pop-up chart is available to be displayed in the organization chart.

Parameters

Parameter	*Description*
Record.*Record_Name*	Specify the name of the derived/work record that contains the data for the pop-up node. You must include the **Record** keyword.

Returns

None.

OrgChart Class Properties

These properties are used by the OrgChart class. The properties are described in alphabetic order.

CenterFocusNode

Description

Specify whether the node with focus displays centered in the chart area.

Value	***Direction***
0	Center the focus node
2	Do not center the focus node

The focus node is not centered if the chart fits in the chart area without a horiztonal scroll, even if CenterFocusNode = 0.

The focus node is not centered if any pop-ups are opened. The display of pop-ups in the visible chart area takes priority over focus node centering.

Note. If multiple pop-ups are opened, the last popup opened is not necessarily set to the visible area. This is determined by the position of the pop-up in the rowset.

The default is 0.

This property is read-write.

ChartScrollType

Description

Specify whether to use scroll bars or alternative scrolling to move the chart within the visible chart area.

Value	***Direction***
0	Use scroll bars
1	Use alternative scrolling. This allows the user to use the scroll navigator or to grab the surface of the organization chart background to move the chart in any direction.

Note. To use the mouse hand feature, the chart background color must be specified in the PSORGCHART style sheet. By default, the background color is white.

Note. On an Apple iPad navigation is done using a finger on a touch screen, so scroll bars, the scroll navigator, and mouse hand features are not displayed on an Apple iPad. For an Apple iPad the behavior is the same for both ChartScrollType = 1 and ChartScrollType = 0.

This property takes an integer value.

The default is 0.

This property is read-write.

Collapsed_Msg

Description

Specify the mouseover text message for the collapsed icon image.

An error message is issued if Collapsed_Msg is set but CollapsedImage is not set.

The default is "Expand".

This property is read-write.

CollapsedImage

Description

Specify the name of the collapsed node image.

If the CollapsedImage and ExpandedImage properties are not specified, then no expanded/collapsed icon appears on the node and the expand/collapse actions are disabled.

This property is read-write.

CollapseMainIconSpace

Description

Specify whether to reduce the space at the top of the node reserved for the main icon..

Value	***Direction***
0	Reserve space at the top of the node for the main icon.
1	Reduce the space at the top of the node reserved for the main icon.

This property takes an integer value.

The default is 0.

This property is read-write.

CrumbDescrStyle

Description

Use this property to specify the style class name that will be used to control the style of linkable breadcrumbs.

The default style class is PT_ORGCHART_BRDCRM.

Oracle recommends that you use the default breadcrumb style.

This property is read-write.

CrumbMaxDisplayLength

Description

Use this property to set the maximum description length that will appear for the breadcrumb.

A breadcrumb description has a maximum of 50 characters in length. If CrumbMaxDisplayLength is set to 30, then only the first 30 characters of the description appear and an ellipsis ("…") is appended.

On mouseover, the whole text of the breadcrumb appears.

The default is to show the full description.

This property takes a number value.

This property is read-write.

CrumbSeparatorImage

Description

Use this property to specify the image name for the image that appears between breadcrumb entries.

If this property is not set, then no image appears between breadcrumb entries. Instead, breadcrumb entries are separated by three spaces.

This property takes a string value.

This property is read-write.

DefaultImage

Description

Use this property to set the default image name for the image that will appear if ImageLocation is not 0 and no image is set in the work record.

This property is read-write.

Direction

Description

Use this property to specify chart node direction.

Value	*Direction*
0	Left to Right
2	Up to Down

For an organization chart, the only valid values are 0 (Left to Right) and 2 (Up to Down).

The default is 2 (Up to Down).

This property is read-write.

DropDownBoxStyle

Description

Specify the style for the drop-down list box header for the organization chart.

Value	*Description*
PT_ACTION_POSITION	The button appears to left, aligned with the other descriptors.
PT_ACTION_POSITION_RIGHT	The button appears to right.
PT_ACTION_POSITION_CENTER	The button appears in the center of the node.

The default style class is PT_ACTION_POSITION.

This property is read-write.

Expanded_Msg

Description

Specify the mouseover text message for the expanded icon image.

An error message is issued if Expanded_Msg is set but ExpandedImage is not set.

The default is "Collapse".

This property is read-write.

ExpandedImage

Description

Specify the name of the expanded node image.

If the CollapsedImage and ExpandedImage properties are not specified, then no expanded/collapsed icon appears on the node and the expand/collapse actions are disabled.

This property is read-write.

HasLegend

Description

Use this property to specify if a legend appears with the chart. This property takes a Boolean value: True if the legend appears with the chart, and False otherwise.

The default value is False.

This property is read-write.

Height

Description

Use this property to specify the height of the chart. This property takes a numeric value. The unit of measurement is pixels.

If you try to read this property before setting it, the value returned is 0.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Height, page 456

ImageHeight

Description

Use this property to specify the height of the node image. This property takes a numeric value. The unit of measurement is pixels.

Note. If ImageHeight is very large relative to the chart, tooltips may not have room to display properly. If this occurs, you need to reduce ImageHeight or increase chart size.

The default is 0.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Height, page 456

ImageLocation

Description

Use this property to specify chart node image location:

Value	***Direction***
0	No image
1	Image on the left
2	Image on the right

The default is 0 (No image).

This property is read-write.

ImageMouseoverMagnificationFactor

Description

Use this property to specify the mouseover magnification factor on the node image. This property takes a numeric value. A value of 100 produces a mouseover image the same size as the node image. If the value is set to 0 or 100, no image magnification will occur.

Note. ImageMouseoverMagnificationFactor is ignored on the Apple iPad. On an Apple iPad the user is able to zoom the entire page to enlarge the image.

Valid values are 0 to 1000.

The default is 0.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Height, page 456

LegendPosition

Description

The OrgChart class only supports %ChartLegend_Top, which is to display the legend on top of the chart.

This property is read-write.

LegendStyle

Description

Use this property to specify the style of the legend. The values for this property are the style classes contained in the style sheet associated with the chart.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_ORGCHART_LEGEND.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

LegendTopSpace

Description

Use this property to set the space between the breadcrumb and legend.

The default is no space between the legend and the breadcrumbs.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

MainTitle

Description

Use this property to specify the text for the main title of the chart.

This property takes a string value.

This property is read-write.

MainTitleStyle

Description

Use this property to specify the style of the main title. The values for this property are the style classes contained in the style sheet associated with the chart.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default value is "PT_ORGCHART_TITLE".

This property is read-write.

MaxDropdownDisplayItem

Description

Specify the maximum number of list items that will appear in the drop down box before the up-and-down arrow vertical scrolling is implemented.

This property takes an integer value.

The default value is 7.

This property is read-write.

MaxPopupDisplayNode

Description

The maximum number of nodes to be displayed in the pop-up before a vertical scrollbar appears in the pop-up.

For instance, if MaxPopupDisplayNode is set to 3 and the pop-up has more than 3 nodes, then the pop-up will only display the first 3 nodes with a scrollbar in the pop-up so the user can scroll down to see the other nodes.

The default value is 3.

This property is read-write.

NodeDescr1Style

Description

The style class name that will be used to control the style of the main chart node descriptor number 1.

Each node can have up to seven descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_ORGNODE_DESC1.

This property is read-write.

NodeDescr2Style

Description

The style class name that will be used to control the style of the main chart node descriptor number 2.

Each node can have up to seven descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_ORGNODE_DESC2.

This property is read-write.

NodeDescr3Style

Description

The style class name that will be used to control the style of the main chart node descriptor number 3.

Each node can have up to seven descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_ORGNODE_DESC3.

This property is read-write.

NodeDescr4Style

Description

The style class name that will be used to control the style of the main chart node descriptor number 4.

Each node can have up to seven descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_ORGNODE_DESC4.

This property is read-write.

NodeDescr5Style

Description

The style class name that will be used to control the style of the main chart node descriptor number 5.

Each node can have up to seven descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_ORGNODE_DESC5.

This property is read-write.

NodeDescr6Style

Description

The style class name that will be used to control the style of the main chart node descriptor number 6.

Each node can have up to seven descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_ORGNODE_DESC6.

This property is read-write.

NodeDescr7Style

Description

The style class name that will be used to control the style of the main chart node descriptor number 7.

Each node can have up to seven descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_ORGNODE_DESC7.

This property is read-write.

NodeMaxDisplayDescLength

Description

Set the maximum descriptor length that will appear in the node.

If the full descriptor text is longer than NodeMaxDisplayDescLength, then an ellipsis ("…") is appended to the displayed text. The entire text appears in a mouseover.

The default value is 50 characters.

This property is read-write.

OptimizeHorizontalSpace

Description

Set this property to automatically maximize or minimize the chart area horizontally.

Value	***Direction***
0	Use the specified size.
1	Expand horizontally to the right edge of the browser page. This option should only be used when there is only blank space to the right of the chart. Otherwise, any objects in that area will be pushed off the viewable area.
2	Contract horizontally to just to the right of the node furthest to the right.
3	Fit the chart area to the chart horizontally.

This property takes an integer value.

The default is 0.

This property is read-write.

OptimizeVerticalSpace

Description

Set this property to automatically maximize or minimize the chart area vertically.

Value	***Direction***
0	Use the specified size.
1	Expand vertically to the right edge of the browser page. This option should only be used when there is only blank space at the bottom of the chart. Otherwise, any objects in that area will be pushed off the viewable area.
2	Contract vertically to just below the bottom of the last visible chart node.
3	Fit the chart area to the chart vertically.

This property takes an integer value.

The default is 0.

This property is read-write.

PopupHeaderStyle

Description

The style class that will be used to control the style of the pop-up header and the header descriptor.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_POPUPNODE_HEADER.

This property is read-write.

PopupNodeDescr1Style

Description

The style class name that will be used to control the style of the pop-up node descriptor number 1.

Each node can have up to eight descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_POPUPNODE_DESC1.

This property is read-write.

PopupNodeDescr2Style

Description

The style class name that will be used to control the style of the pop-up node descriptor number 2.

Each node can have up to eight descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_POPUPNODE_DESC2.

This property is read-write.

PopupNodeDescr3Style

Description

The style class name that will be used to control the style of the pop-up node descriptor number 3.

Each node can have up to eight descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_POPUPNODE_DESC3.

This property is read-write.

PopupNodeDescr4Style

Description

The style class name that will be used to control the style of the pop-up node descriptor number 4.

Each node can have up to eight descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_POPUPNODE_DESC4.

This property is read-write.

PopupNodeDescr5Style

Description

The style class name that will be used to control the style of the pop-up node descriptor number 5.

Each node can have up to eight descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_POPUPNODE_DESC5.

This property is read-write.

PopupNodeDescr6Style

Description

The style class name that will be used to control the style of the pop-up node descriptor number 6.

Each node can have up to eight descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_POPUPNODE_DESC6.

This property is read-write.

PopupNodeDescr7Style

Description

The style class name that will be used to control the style of the pop-up node descriptor number 7.

Each node can have up to eight descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_POPUPNODE_DESC7.

This property is read-write.

PopupNodeDescr8Style

Description

The style class name that will be used to control the style of the pop-up node descriptor number 8.

Each node can have up to eight descriptors, each with its own style.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_POPUPNODE_DESC8.

This property is read-write.

Style

Description

A *style sheet* is a definition, like a record or field definition, that you create in PeopleSoft Application Designer. It contains a collection of formatting styles, called *classes,* each of which can be applied to either the entire chart or to specific elements within a chart.

Use this property to specify the overall style of the chart by specifying a style class. The value must be a valid class within the style sheet specified for the chart.

Individual style properties (such as Legend Style, MainTitleStyle, and so on) can be used to override this value for individual elements of the chart.

The default style class is PT_ORGCHART.

This property is read-write.

SuppressPeopleCodeOnNode

Description

Specify whether to suppress processing, such as re-centering the chart, when a user clicks on a chart node if no PeopleCode is associated with the node.

This property takes a Boolean value: True to suppress a click on a chart node, and False otherwise.

The default value is False.

This property is read-write.

SuppressPeopleCodeOnImage

Description

Specify whether to suppress a processing, such as re-centering the chart, when a user clicks on a chart image if no PeopleCode is associated with the image.

This property takes a Boolean value: True to suppress a click on a chart image, and False otherwise.

The default value is False.

This property is read-write.

UnlinkCrumbDescrStyle

Description

Use this property to specify the style class name that will be used to control the style of the unlinkable breadcrumb.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_ORGCHART_UNLINK_BRDCRM.

Oracle recommends that you use the default breadcrumb unlinkable style.

This property is read-write.

VerticalSpace

Description

Use this property to specify the amount of space above the first level of nodes in an organization chart. If the chart has a legend, the vertical space is measured as the distance from the top of the first level of nodes to the bottom of the legend. If the chart has no legend, the vertical space is the distance from the top of the first level of nodes to the top of the chart area.

This property only applies to charts with a vertical orientation (Direction = 2).

This property takes a numeric value. The unit of measurement is pixels.

The default value is 10 pixels.

This property is read-write.

Width

Description

Use this property to specify the width of the chart. This property takes a numeric value. The unit of measurement is pixels.

If you try to read this property before setting it, the value returned is 0.

This property is read-write.

RatingBoxChart Class Methods

These methods are used by the RatingBoxChart class. The methods are described in alphabetic order.

SetLegend

Syntax

SetLegend*(&Array_of_String)*

Description

Use the SetLegend method to specify legend text. Legend text labels the images set in the RatingBoxChart class method SetLegendImg.

Note. Legend text is not automatically translated. Be sure to use translated text, such as message catalog entries.

Parameters

Parameter	*Description*
&Array_of_String	Specify an already instantiated array of string, containing the text that you want to use for the legend.

Returns

None.

Example

```
&LegendArray = CreateArray("A","B", C","D");
&oRatingBoxChart.SetLegend(&LegendArray);
```

See Also

Chapter 11, "Charting Classes," SetDataSeries, page 443

Chapter 8, "Array Class," page 257

Chapter 11, "Charting Classes," SetLegendImg, page 544

SetLegendImg

Syntax

SetLegendImg*(&Array_of_String)*

Description

Use the SetLegendImg method to specify the legend image names.

Parameters

Parameter	*Description*
&Array_of_String	An already instantiated array containing the names of the images that you want to use with the legend.

Returns

None.

SetRBNodeData

Syntax

SetRBNodeData*(&Rowset)*

Description

Use the SetRBNodeData method to specify the level 1 component rowset that is the source for the node data for the rating box chart.

If you make a change to the underlying data of a chart, call the SetRBNodeData method again to update the chart.

Parameters

Parameter	*Description*
&Rowset	An already instantiated and populated component rowset object that references the rating chart node record.

Returns

None.

SetRBNodeRecord

Syntax

SetRBNodeRecord(**Record.***RecordName*)

Description

Use the SetRBNodeRecord method to specify the record that is to contain the node data for the rating box chart.

Parameters

Parameter	*Description*
Record.*RecordName*	Specify the name of the record to be used as the node record.

Returns

None.

SetXAxisLabels

Syntax

SetXAxisLabels*(&Array_of_Any)*

Description

Use the SetXAxisLabels method to specify an array of labels for the X axis.

Parameters

Parameter	*Description*
&Array_of_Any	Specify an already instantiated array of any that contain the labels that you want to use for the X Axis. The array size must match the value in the XAxisBoxNum property. If they do not match, the system throws a runtime error.

Returns

None.

See Also

Chapter 8, "Array Class," page 257

SetYAxisLabels

Syntax

`SetYAxisLabels`*`(&Array_of_Any)`*

Description

Use the SetYAxisLabels method to specify an array of labels for the Y axis.

Parameters

Parameter	*Description*
&Array_of_Any	Specify an already instantiated array of any that contain the labels that you want to use for the Y Axis. The array size must match the value in the YAxisBoxNum property. If they do not match, the system throws a runtime error.

Returns

None.

See Also

Chapter 8, "Array Class," page 257

RatingBoxChart Class Properties

These properties are used by the RatingBoxChart class. The properties are described in alphabetic order.

BoxMaxDisplayItems

Description

Specify how many nodes are to be shown in the displayable area of the box. When the number of nodes to be shown exceeds BoxMaxDisplayItems, a link labeled "View all (N)" appears at the bottom right of the box, where *N* is the total number of nodes in the box. The user can click the link to launch a pop-up that contains all the nodes. A scrollbar enables the user to scroll through all the nodes in the list.

Only one pop-up window will appear in the rating box chart, so when the user clicks another "View all *(N)*" link in another box, the previous pop-up window closes and the new pop-up window opens at the center of the rating box.

If an invalid (negative) value is specified, an error will be thrown at runtime.

The default value is 1.

This property is read-write.

DraggedNodeStyle

Description

Use this property to specify the style class that controls the appearance of a node as it is being dragged.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_DRAGGED_NODE.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

GridLineType

Description

Use this property to specify the style of the grid lines.

The default value is %ChartLine_Solid. The possible values are:

Value	***Description***
%ChartLine_Solid	Draw grid lines with solid lines.
%ChartLine_Dash	Draw grid lines with lines composed of dashes.
%ChartLine_Dot	Draw grid lines with lines composed of dots.

This property is read-write.

HasLegend

Description

Use this property to specify if a legend appears with the chart. This property takes a Boolean value: True if the legend appears with the chart, and False otherwise.

The default value is False.

This property is read-write.

Height

Description

Use this property to specify the height of the chart. This property takes a numeric value. The unit of measurement is pixels.

The default value is the height of the chart control on the page definition.

If you set a chart height value that cannot fit the X axis and label areas of the chart, a runtime error is thrown and the chart is not rendered.

If you set a negative value, a runtime error is thrown and the chart is not rendered.

If you try to read this property before setting it, the value returned is 0.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Height, page 456

IconOnlySelectedQuadrantStyle

Description

Use this property to specify the style class that controls the appearance of border properties of the selected quadrant of an icon-only rating box chart when the system displays the view all pop-up.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default is PT_RATBOX_ICONONLY_BOX.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

IsDragable

Description

Specify whether the node in the rating box can be dragged and dropped. If the property is specified as *True*, then user will be able to drag the node in the rating box and drop it to another quadrant. If the property is *False*, then the drag and drop action is disabled.

The default value is *True.*

This property is read-write.

See Also

Chapter 11, "Charting Classes," Height, page 456

LegendPosition

Description

Use this property to specify where the legend should appear in relationship to the chart. You can specify either a numeric or constant value for this property.

The values are:

Numeric Value	Constant Value	Description
2	%ChartLegend_Top	Display legend at the top of the chart.
3	%ChartLegend_Bottom	Display legend below the chart.

If you set a value other than *2* or *3*, a runtime error is thrown and the chart is not rendered.

This property is read-write.

MainTitle

Description

Use this property to specify the text for the main title of the chart.

This property takes a string value.

This property is read-write.

MainTitleStyle

Description

Use this property to specify the style of the main title. The values for this property are the style classes contained in the style sheet associated with the chart.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_TITLE.

This property is read-write.

NDMaxDisplayDescLength

Description

Specify the maximum number of characters that will display for the description.

When the ShowNodeDescription chart property is set to True, then both the icon and the description of the node will appear in the quadrant.

If the description is larger than NDMaxDisplayDescLength then the node displays the number of characters specified in NDMaxDisplayDescLength, followed by an ellipsis.

If the full node description or the number of characters specified in NDMaxDisplayDescLength cannot fit in a box at runtime the description is automatically truncated. An ellipsis (...) follows the truncated text.

You only need to set this property if you want the truncated text to be less than the automatically truncated text.

When the user mouses over the truncated node description, the whole text displays in a tool tip.

If an invalid (negative) value is specified, an error will be thrown at runtime.

The default value is 50.

This property is read-write.

PopUpHeaderStyle

Description

Use this property to specify the style class that controls the appearance of the pop-up header.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_POPUP_HEADER.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

PopUpHeight

Description

Specify the height of the pop-up chart in pixels.

This parameter takes a number value.

The recommended value is half of the rating box chart height.

If you set a pop-up chart height value that cannot fit the X axis area of the popup chart, a runtime error is thrown.

The default value is 0. If a negative value is specified, an error will be thrown at runtime.

This property is read-write.

PopUpStyle

Description

Use this property to specify the style class that controls the appearance of the pop-up.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_POPUP.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

PopUpWidth

Description

Specify the width of the pop-up chart in pixels.

This parameter takes a number value.

The recommended value is half of the rating box chart width.

If you set a pop-up chart width value that cannot fit the Y axis area of the popup chart, a runtime error is thrown.

The default value is 0. If a negative value is specified, an error will be thrown at runtime.

This property is read-write.

SelectedQuadrantStyle

Description

Use this property to specify the style class that controls the appearance of the quadrant when it is selected.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_SELECTED_BOX.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

ShowNodeDescription

Description

Specify whether the chart should show the node description. This property takes a Boolean value.

Value	***Description***
True	Both the icon and description for the node appear.
False	Only the icon for the node appears.

The default value is True.

This property is read-write.

Style

Description

Use this property to specify the overall style for the chart by specifying a style class. The value must be a valid class within the style sheet specified for the chart.

The default style class is PT_RATBOX_CHART.

This property is read-write.

ViewAllStyle

Description

Use this property to specify the style class that controls the appearance of the View All link.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_VIEWALL_DESCR.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

Width

Description

Use this property to specify the width of the chart. This property takes a numeric value. The unit of measurement is pixels.

The default value is the width of the chart control on the page definition.

If you set a chart width value that cannot fit the Y axis and label areas of the chart, a runtime error is thrown.

If a negative value is specified, an error will be thrown at runtime and the chart will not be rendered.

If you try to read this property before setting it, the value returned is 0.

This property is read-write.

XAxisBoxNum

Description

Specify the number of boxes in the X axis.

This property is required. The value must be equal to or greater than 1.

If a value less than 1 is specified, an error will be thrown at runtime.

This property is read-write.

XAxisLabelStyle

Description

Use this property to specify the style of the X-axis label. The values for this property are the style classes contained in the style sheet associated with the chart.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_XAXIS.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

XAxisTitle

Description

Use this property to specify the text for the X-axis title.

This property is read-write.

XAxisTitleStyle

Description

Use this property to specify the style of the X-axis title. The values for this property are the style classes contained in the style sheet associated with the chart.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_XTTL.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

YAxisBoxNum

Description

Specify the number of boxes in the Y axis.

This property is required. The value must be equal to or greater than 1.

If a value less than 1 is specified, an error will be thrown at runtime.

This property is read-write.

YAxisLabelStyle

Description

Use this property to specify the style of the Y-axis label. The values for this property are the style classes contained in the style sheet associated with the chart.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_YAXS.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

YAxisTitle

Description

Use this property to specify text of the title for the Y axis.

This property is read-write.

YAxisTitleStyle

Description

Use this property to specify the style of the Y-axis title. The values for this property are the style classes contained in the style sheet associated with the chart.

The values for this property are controlled by the specified style class in the style sheet associated with the chart.

The default style class is PT_RATBOX_YTITL.

This property is read-write.

See Also

Chapter 11, "Charting Classes," Style Sheets, page 383

Charting Examples

The following are some example of charts, with the PeopleCode used to create them.

Creating a Chart Using the Chart Class

This section demonstrates how to create a simple chart based on the Chart class. This chart is meant to be used to demonstrate basic principles of creating charts and give an opportunity to experiment with different chart types. It is not typical of how a chart is used in PeopleSoft applications.

The first six steps also apply to creating charts using the Gantt class, the OrgChart class, and the RatingBoxChart class.

The steps to create a chart are:

1. Place a chart control on a page.
2. Associate the chart control with a record field.
3. Create the chart data record.
4. Add PeopleCode to instantiate a chart object.
5. Set the data source.
6. View the page in the browser.
7. Add data for the chart.
8. Specify the series data.
9. Specify chart colors.
10. Add text elements.
11. Review the complete program.

Placing a Chart Control on a Page

In Application Designer, create a new page.

Select Insert, Chart and draw a rectangle to represent the area the chart will occupy. The chart can be any size, and you can have more than one chart on a page.

Place the page on a component and register the component.

Associating the Chart Control with a Record Field

Each chart on a component must be associated with a unique field. Double-click the chart control to access the Chart Properties and enter a record name and field name.

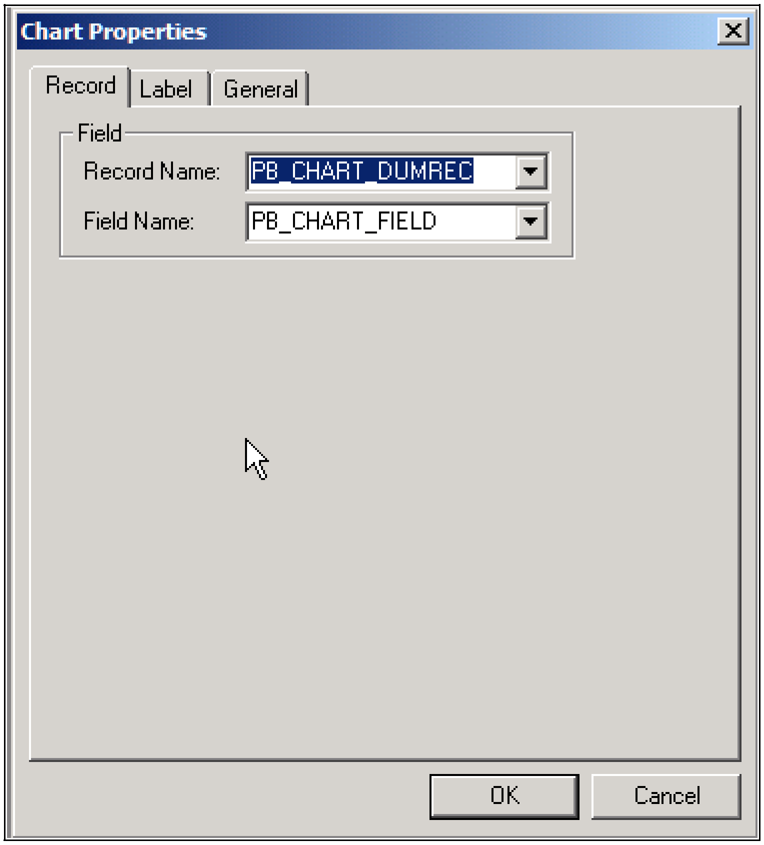


Chart properties dialog

The chart field has no special requirements. Its only purpose is to provide a reference to the field in PeopleCode.

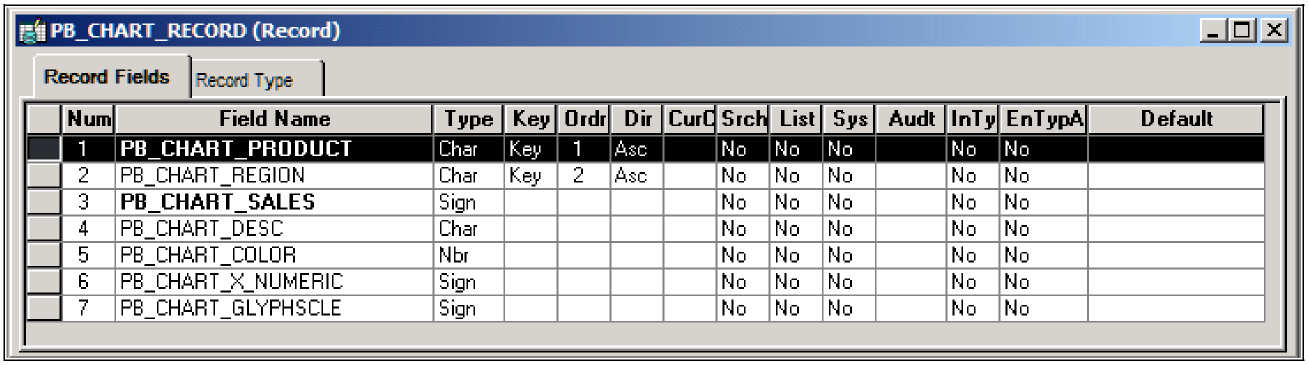
This sample chart uses a derived/work record named SC_CHART_DUMREC that is dedicated to this purpose.

Creating the Chart Data Record

The chart needs either a record or a rowset to serve as the data source. This sample chart uses a record. See the Chart class SetData method for details about using a rowset.

See Chapter 11, "Charting Classes," SetData, page 435.

At a minimum, the record must have fields for the X-axis data and the Y-axis data. Optionally, the record can have fields for series data, chart color, and glyph size.



PB_CHART_RECORD (Record)

Record Fields | Record Type

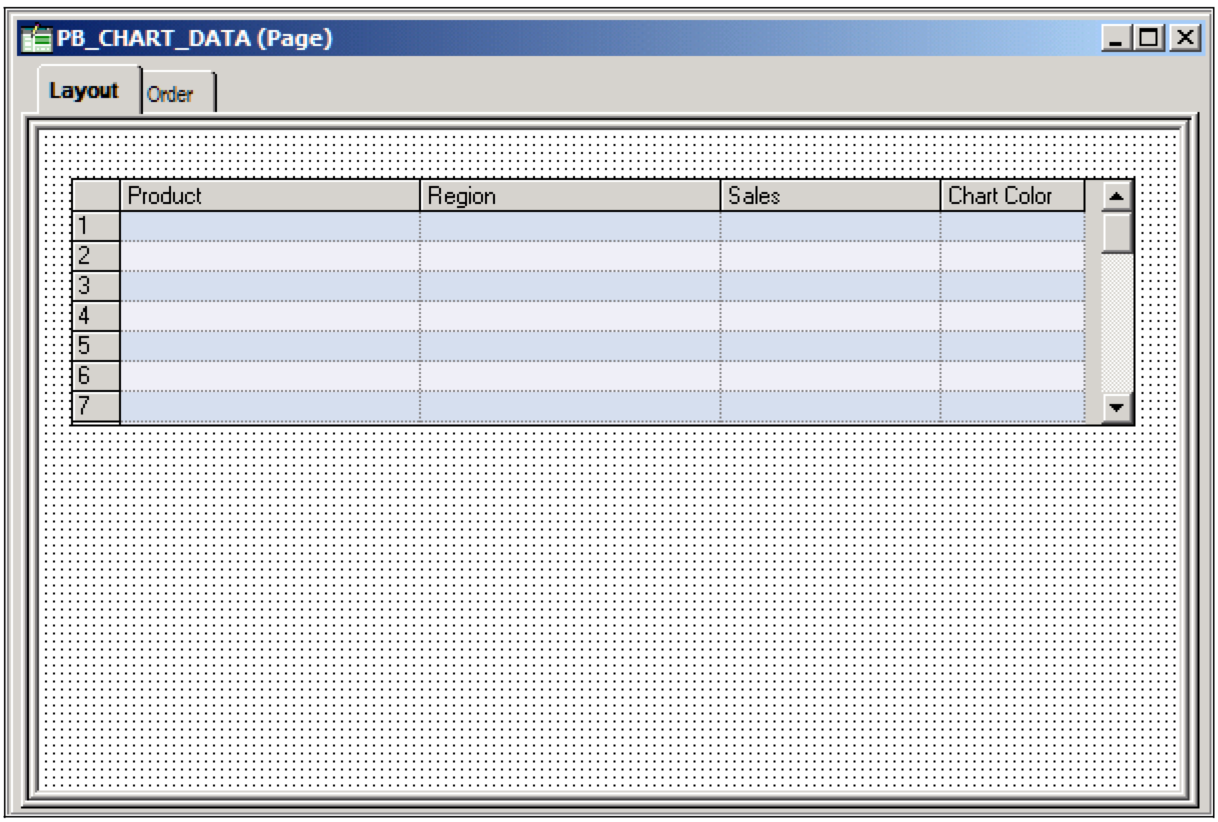
	Num	Field Name	Type	Key	Ordr	Dir	CurC	Srch	List	Sys	Audt	InTy	EnTypA	Default
	1	PB_CHART_PRODUCT	Char	Key	1	Asc		No	No	No		No	No	
	2	PB_CHART_REGION	Char	Key	2	Asc		No	No	No		No	No	
	3	PB_CHART_SALES	Sign					No	No	No		No	No	
	4	PB_CHART_DESC	Char					No	No	No		No	No	
	5	PB_CHART_COLOR	Nbr					No	No	No		No	No	
	6	PB_CHART_X_NUMERIC	Sign					No	No	No		No	No	
	7	PB_CHART_GLYPHSCLE	Sign					No	No	No		No	No	

Example of a chart data record

The record must be available in the component buffer at runtime. You can use a database record that is already part of your application or you can use a derived/work record or a component rowset that you populate at runtime. For greatest control, you will normally populate your record or rowset at runtime.

For the sample chart, place the record on a grid on a second page on the sample component. This placement gives you the opportunity to experiment with the data.

Your chart data record must be placed at level 1 on the same component as the chart so that the data is available in the component buffer at runtime. In your application, you can hide the grids or scroll areas that hold the records. For the sample chart, you will keep the grid visible so you can enter sample data.



Example of a page definition with chart data grid

Adding PeopleCode to Instantiate a Chart Object

This example instantiates a chart object using the Chart class, so it can be used to create bar charts, line charts, pie charts, and so on. Other examples in this section demonstrate how to create charts based on the Gantt class, OrgChart class, and RatingBoxChart class.

See Chapter 11, "Charting Classes," Creating a Gantt Chart, page 585; Chapter 11, "Charting Classes," Creating an Organization Chart, page 588 and Chapter 11, "Charting Classes," Creating a Rating Box Chart, page 592.

Add this PeopleCode to the Activate event on the page:

```
/* Declare and instantiate a chart object */
Component Chart &cChart;

&cChart = GetChart(SC_CHART_DUMREC.SC_CHART_FIELD);
```

Chart PeopleCode is commonly placed on the Activate event, but it can be placed on any appropriate event. You could, for instance, execute the chart PeopleCode from a FieldChange event associated with a push button.

Setting the Data Source

The SetData method takes either a record or a rowset as an argument. In general, specify a record or component rowset when building a chart from page data and a standalone rowset when building a chart using an iScript.

In this example, the PeopleCode sets Product as the series data and sets Region to the X axis, so that in a bar chart products will be grouped along the X axis by region. In a line chart, each series is represented by a line. Sales figures are charted on the Y axis.

Note. SetData, SetDataXAxis, and SetDataYAxis are Chart class methods. The other charting classes provide equivalent methods to specify data sources.

```
/* Set the chart data record and specify X-axis, Y-axis, and series data */

&cChart.SetData(Record.SC_CHART_RECORD);
&cChart.SetDataXAxis(SC_CHART_RECORD.SC_REGION);
&cChart.SetDataYAxis(SC_CHART_RECORD.SC_SALES);
```

See Chapter 11, "Charting Classes," SetData, page 435; Chapter 11, "Charting Classes," SetDataSeries, page 443; Chapter 11, "Charting Classes," SetDataXAxis, page 444; Chapter 11, "Charting Classes," SetDataYAxis, page 446 and Chapter 11, "Charting Classes," Creating a Chart Using an iScript, page 596.

Viewing the Page in the Browser

This is the most basic PeopleSoft Chart class chart. It has no data and only the default Y axis.

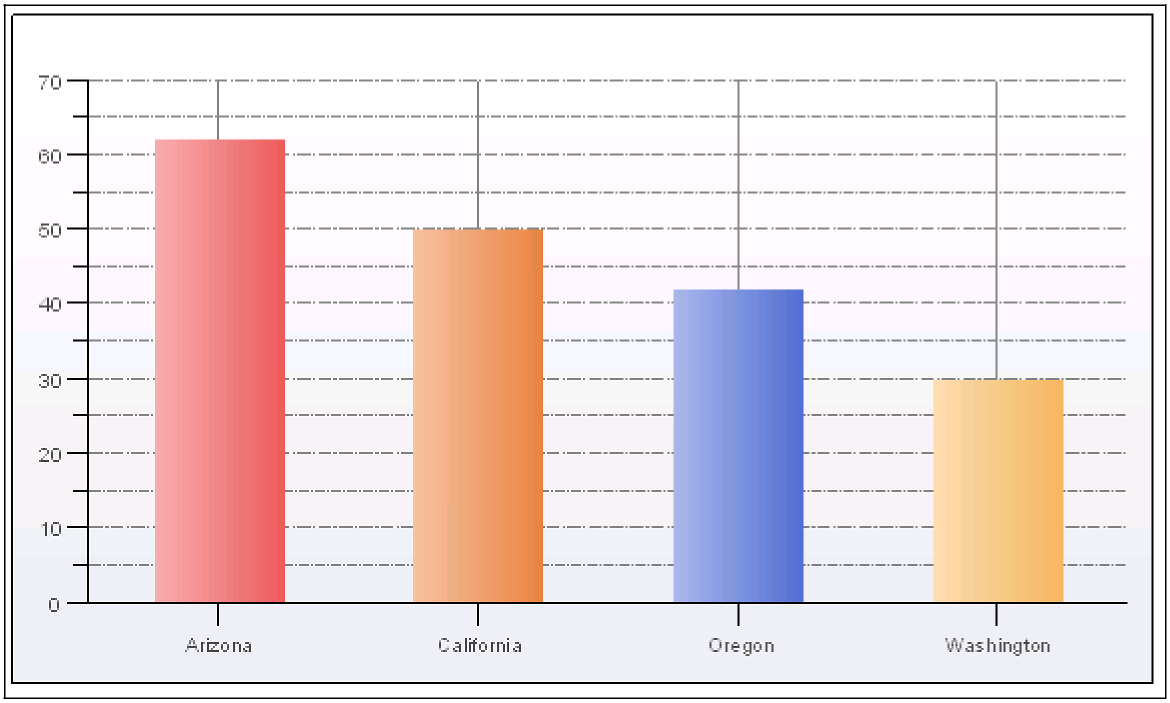
Chart class chart with no data

Adding Data for the Chart

Add the following data to the chart:

Region	***Sales***
Arizona	60
California	50
Oregon	42
Washington	30

This bar chart is generated by the example code and data:



Default Chart class bar chart

This chart graphs sales on the Y axis (the vertical axis) and regions on the X axis (the horizontal axis).

When the data for the chart is grouped according to values, a set of data representing a single value is called a *series*. For instance, if a chart shows sales for different products in each region, then the set of data for each product is a series.

At this point, the chart has only one series, so you do not need to set the series record field.

Specify the Series Data

If you want to graph more than one product on your chart, you need to specify the source of the series data.

For example, add the products shown in this table to your data:

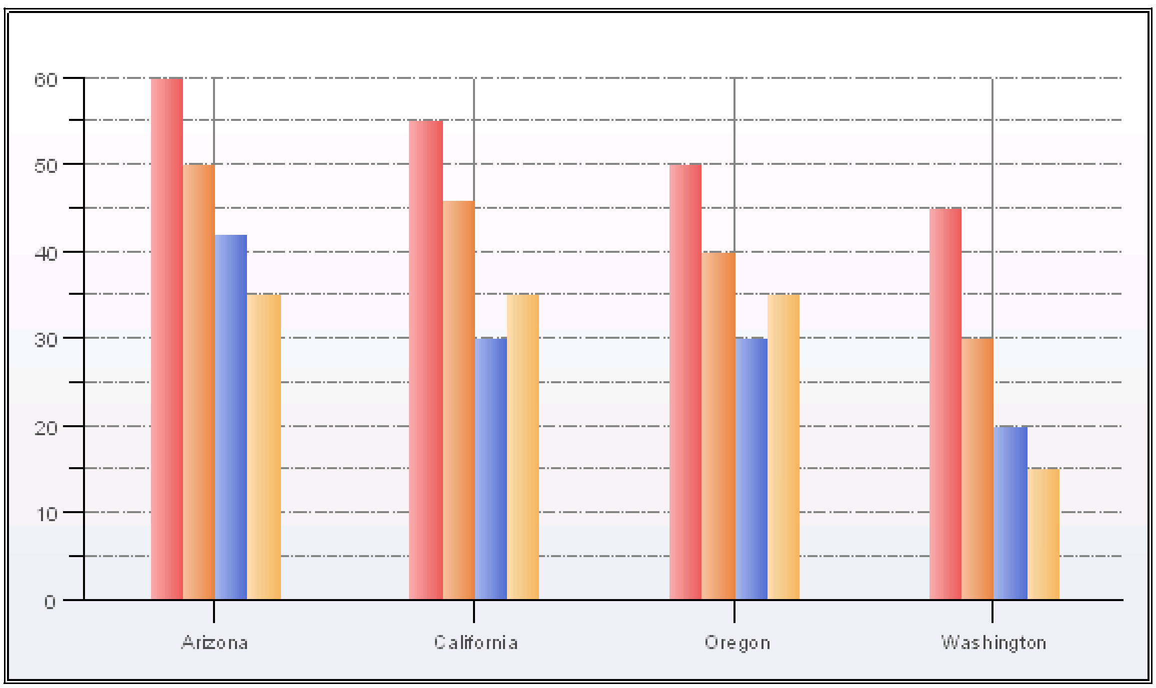
Product	***Region***	***Sales***
Footballs	Arizona	60
Rackets	Arizona	50
Shoes	Arizona	42
Tents	Arizona	35
Footballs	California	55
Rackets	California	46
Shoes	California	30

Product	***Region***	***Sales***
Tents	California	35
Footballs	Oregon	50
Rackets	Oregon	40
Shoes	Oregon	30
Tents	Oregon	25
Footballs	Washington	45
Rackets	Washington	30
Shoes	Washington	20
Tents	Washington	15

Add this PeopleCode to set the series record field:

```
&cChart.SetDataSeries(SC_CHART_RECORD.SC_PRODUCT);
```

The resulting chart is a default 2D bar chart with bars for each region grouped by series:



Default 2D bar chart with products as series

The chart has four series:

- Footballs

- Rackets
- Shoes
- Tents

Each series shows four regions along the X axis:

- Arizona
- California
- Oregon
- Washington

Sales values appear against the Y axis as bar height.

The chart uses the default colors.

Specifying Chart Colors

If you do not want to use the default colors, you need a column containing the colors for each bar. Typically, the bars in a series are colored alike.

Product	***Region***	***Sales***	***Chart Color***
Footballs	Arizona	60	1
Rackets	Arizona	50	13
Shoes	Arizona	42	5
Tents	Arizona	35	15
Footballs	California	55	1
Rackets	California	46	13
Shoes	California	30	5
Tents	California	35	15
Footballs	Oregon	50	1
Rackets	Oregon	40	13
Shoes	Oregon	30	5
Tents	Oregon	25	15
Footballs	Washington	45	1

Product	***Region***	***Sales***	***Chart Color***
Rackets	Washington	30	13
Shoes	Washington	20	5
Tents	Washington	15	15

Add the following PeopleCode to your program to set the colors for your chart:

```
/* Associate color field with chart */
&cChart.SetDataColor(SC_CHART_RECORD.QE_CHART_COLOR);
```

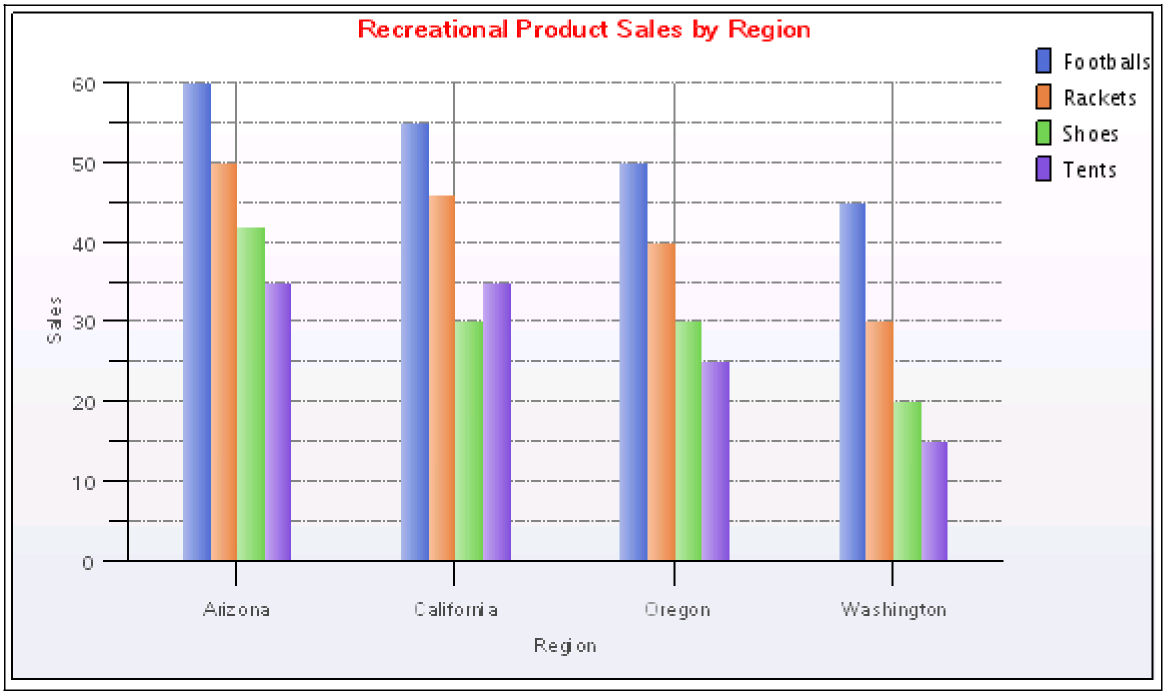
Adding Text Elements

The following PeopleCode adds these elements to your chart:

- Title
- Legend
- Y-axis title
- X-axis title

```
/* Set title and legend properties */
&cChart.MainTitle = "Recreational Product Sales by Region";
&cChart.HasLegend = True;
&cChart.LegendPosition = %ChartLegend_Right;

/* Set axis text properties */
&cChart.XAxisLabelOrient = %ChartText_Horizontal;
&cChart.XAxisTitle = "Region";
&cChart.YAxisTitle = "Sales";
```

This is the resulting chart:



Sample bar chart with custom colors, a main title, axis titles, and a legend

Reviewing the Complete Program

The PeopleCode for this bar chart has been presented in snippets. Here is the complete program that produced the chart:

```
/* Declare a chart object */

Component Chart &cChart;

/* Instantiate the chart object and associate the chart
** with the chart page control */

&cChart = GetChart(SC_CHART_DUMREC.QE_CHART_FIELD);

/* Set the chart data record and specify X-axis, Y-axis,
** and series data */
&cChart.SetData(Record.SC_CHART_RECORD);
&cChart.SetDataSeries(SC_CHART_RECORD.SC_PRODUCT);
&cChart.SetDataXAxis(SC_CHART_RECORD.SC_REGION);
&cChart.SetDataYAxis(SC_CHART_RECORD.SC_SALES);

/* Associate color field with chart */
&cChart.SetDataColor(SC_CHART_RECORD.QE_CHART_COLOR);

/* Set title and legend properties */
&cChart.MainTitle = "Recreational Product Sales by Region";
&cChart.HasLegend = True;
&cChart.LegendPosition = %ChartLegend_Right;
/* Set axis text properties */
&cChart.XAxisLabelOrient = %ChartText_Horizontal;
&cChart.XAxisTitle = "Region";
&cChart.YAxisTitle = "Sales ";
```

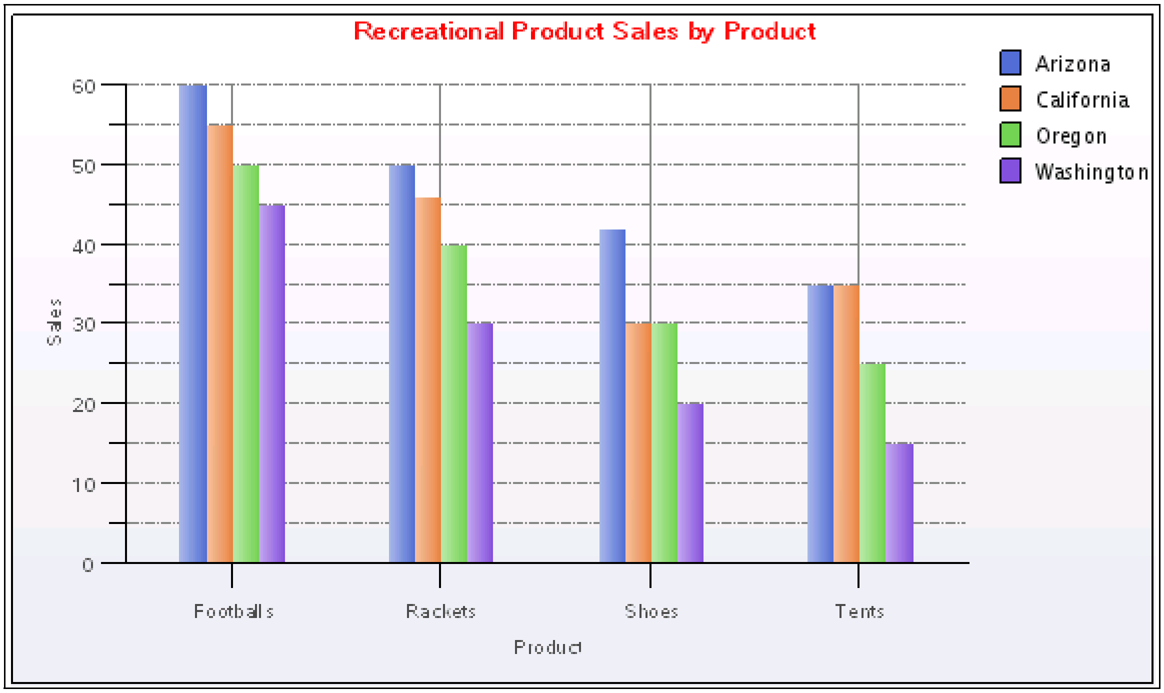

Other Modifications

The remainder of this section shows the effects of modifying other aspects of your chart.

Changing X-Axis and Series Values

To group sales by product instead of by region, switch the series and the X axis. You will also need to reorder colors to correspond to regions instead of products.

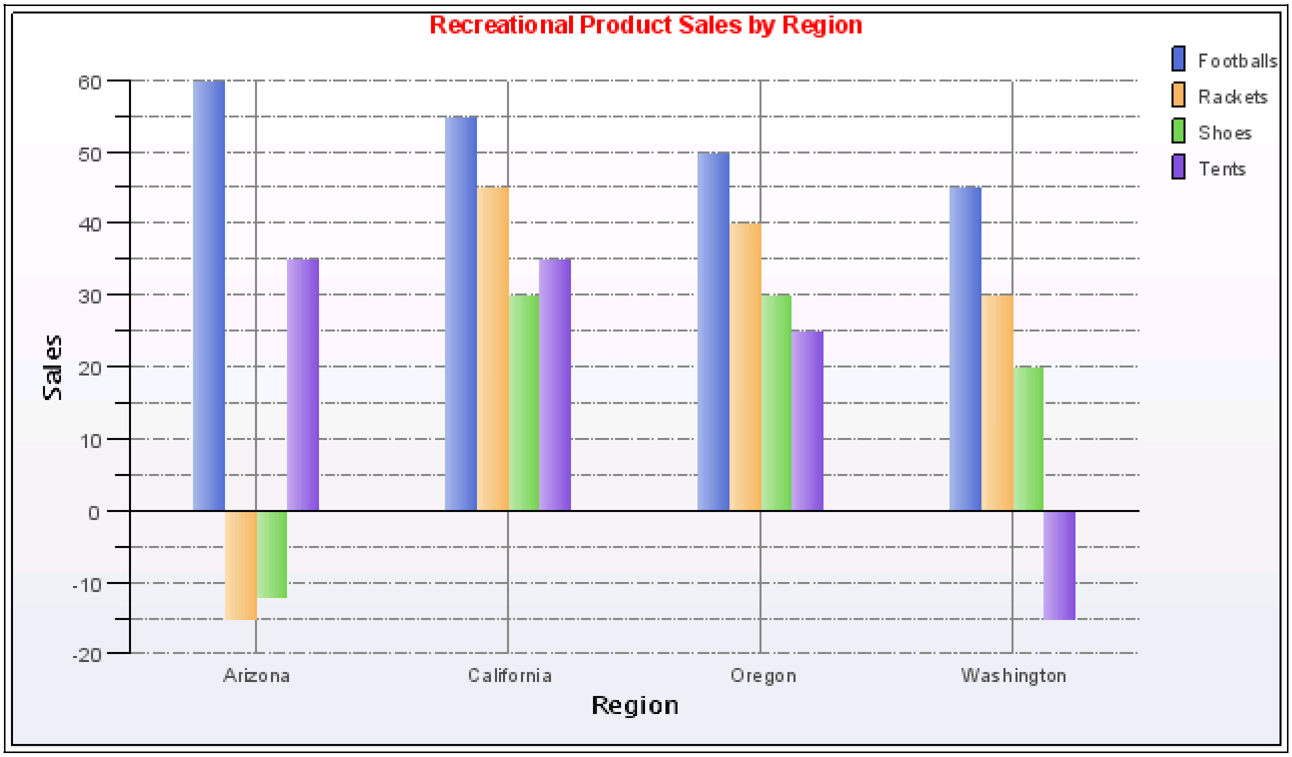
```
&cChart.SetDataSeries(SC_CHART_RECORD.SC_REGION);
&cChart.SetDataXAxis(SC_CHART_RECORD.SC_PRODUCT );

&cChart.XAxisTitle = "Region";
&cChart.MainTitle = "Recreational Product Sales by Product";
```



Bar chart with series set to region

Axis Properties

If your chart has negative values, the bars automatically extend below the X axis:



Bar chart with negative values

If you want to remove the tick marks from the X or Y axis, you need to specify an empty array for the SetXAxisLabels or SetYAxisLabels methods.

The following example removes the tick marks from the Y axis:

```
&Arr = CreateArray("");
&Chart.SetYAxisLabel(&Arr);
```

If you set the X-axis labels to an empty array to remove the ticks, you must be plotting a single series on the chart because labels are automatically generated for each series if you do not provide them.

See Chapter 11, "Charting Classes," SetXAxisLabels, page 452 and Chapter 11, "Charting Classes," SetYAxisLabels, page 452.

You can also use the XAxisCross, XAxisMin, XAxisMax, YAxisMin, YAxisMax, and other axis properties to control the appearance of the axes.

See Chapter 11, "Charting Classes," XAxisCross, page 466; Chapter 11, "Charting Classes," XAxisLabelOrient, page 468; Chapter 11, "Charting Classes," XAxisStyle, page 470; Chapter 11, "Charting Classes," XAxisTicks, page 470; Chapter 11, "Charting Classes," YAxisLabelOrient, page 472; Chapter 11, "Charting Classes," YAxisMax, page 473; Chapter 11, "Charting Classes," YAxisMin, page 473; Chapter 11, "Charting Classes," YAxisStyle, page 474 and Chapter 11, "Charting Classes," YAxisTicks, page 475.

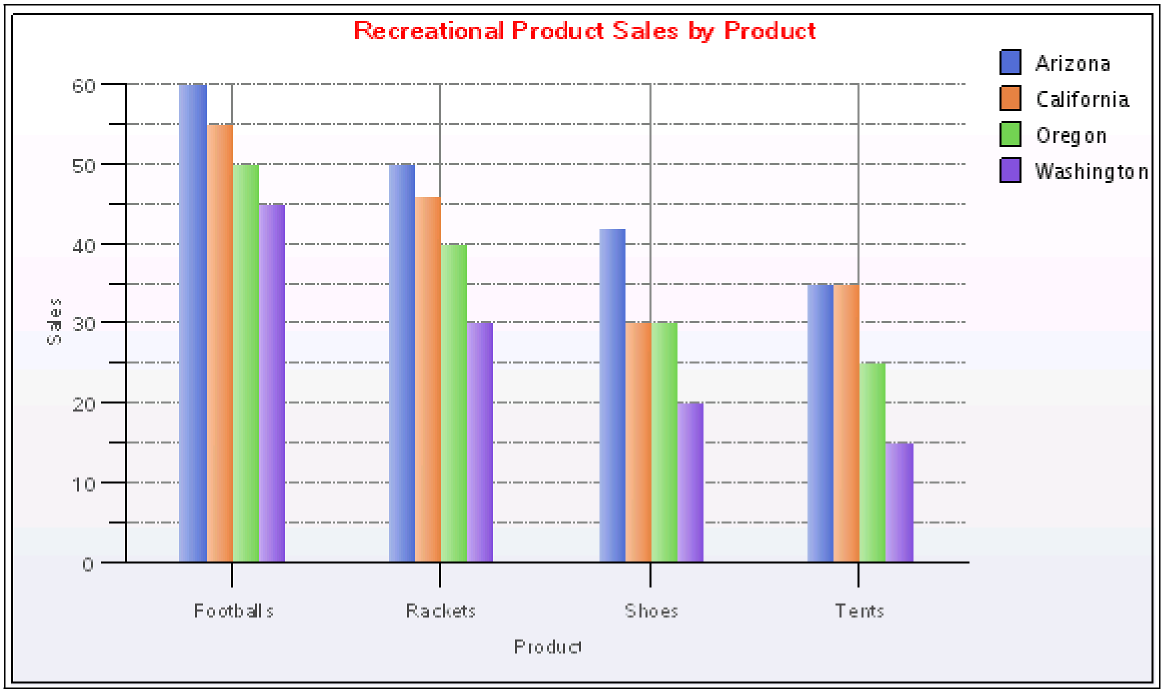
Chart Class Chart Types

Use the Type property to set the chart type.

2D Bar Chart

The default Chart class chart type is a bar chart.

```
&cChart.Type=%ChartType_2DBar;
```



2D bar chart

The following examples show the various chart types using the same data. Each example gives the PeopleCode snippet that creates that chart type.

For a complete list of all Chart class chart types, see Chart class Type property.

See Chapter 11, "Charting Classes," Type, page 465.

The Charting Examples section provides other examples of PeopleCode for charts using Gantt class, OrgChart class, and RatingBoxChart class at the end of the chapter.

See Chapter 11, "Charting Classes," Creating a Gantt Chart, page 585; Chapter 11, "Charting Classes," Creating an Organization Chart, page 588 and Chapter 11, "Charting Classes," Creating a Rating Box Chart, page 592.

3D Bar Chart

A visual 3D effect can add interest to a chart.

Add this code to make your bar chart a 3D bar chart:

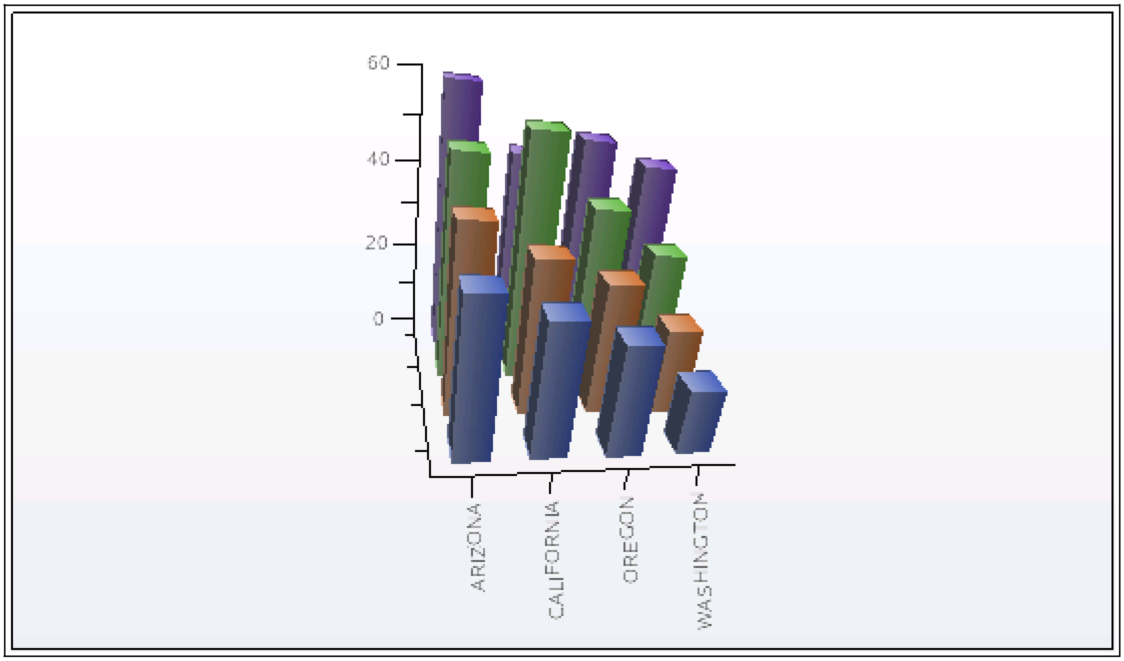
```
&cChart.Type = %ChartType_3DBar;
&cChart.XAxisLabelOrient = 90;
```

This PeopleCode also turns the X-axis labels 90 degrees for better readability. Valid values for XAxisLabelOrient are 0 and 90.

Use the XRotationAngle, YRotationAngle, and ZRotationAngle properties to rotate the bar chart. This example shows the default values for rotation.

The default values are:

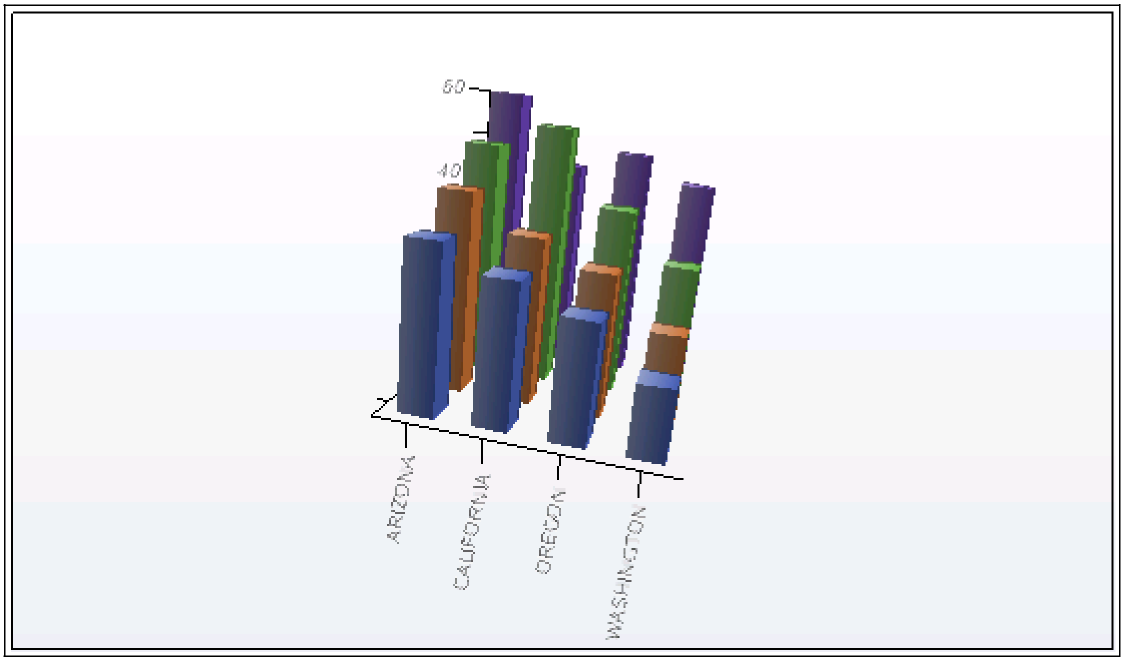
Property	***Value***
XRotationAngle	330
YRotationAngle	345
ZRotationAngle	0



3D bar with default rotation angles

The following 3D bar chart uses these rotation angles:

```
&cChart.XRotationAngle = "340";
&cChart.YRotationAngle = "10";
&cChart.ZRotationAngle = "10";
```



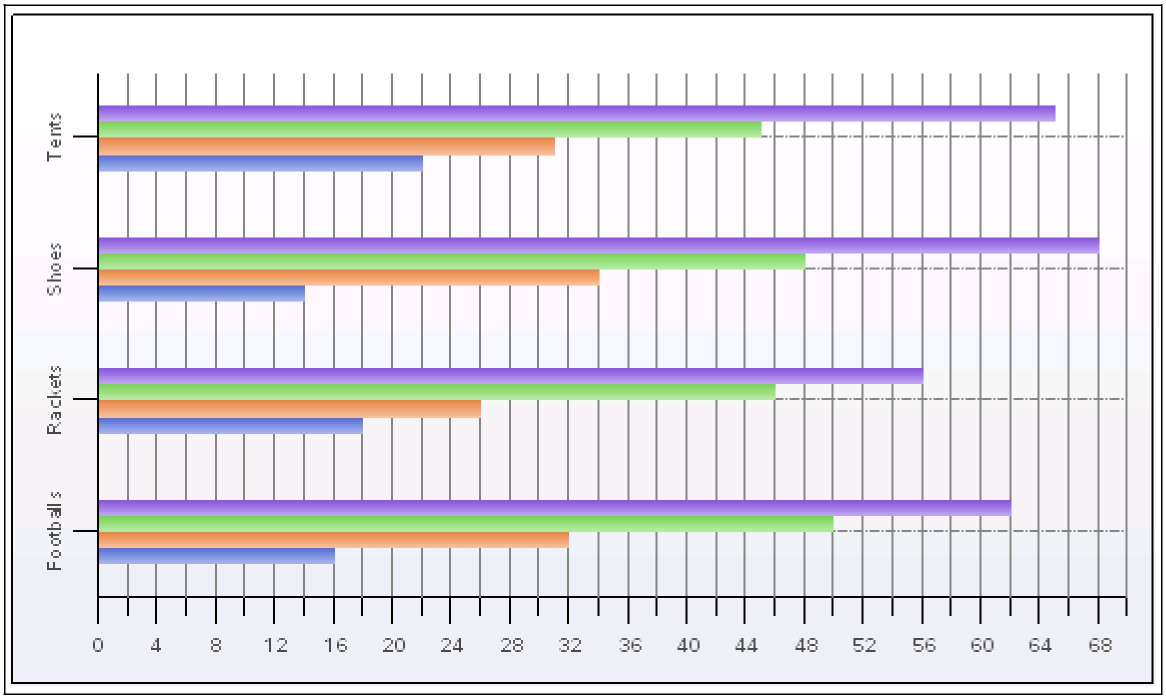
3D Bar chart with modified-axis rotation angles

See Chapter 11, "Charting Classes," XRotationAngle, page 471; Chapter 11, "Charting Classes," YRotationAngle, page 476; Chapter 11, "Charting Classes," ZRotationAngle, page 476 and Chapter 11, "Charting Classes," XAxisLabelOrient, page 468.

Horizontal Bar Chart

Use this line of code to specify a horizontal bar chart:

```
&cChart.Type = %ChartType_2DHorizBar;
```



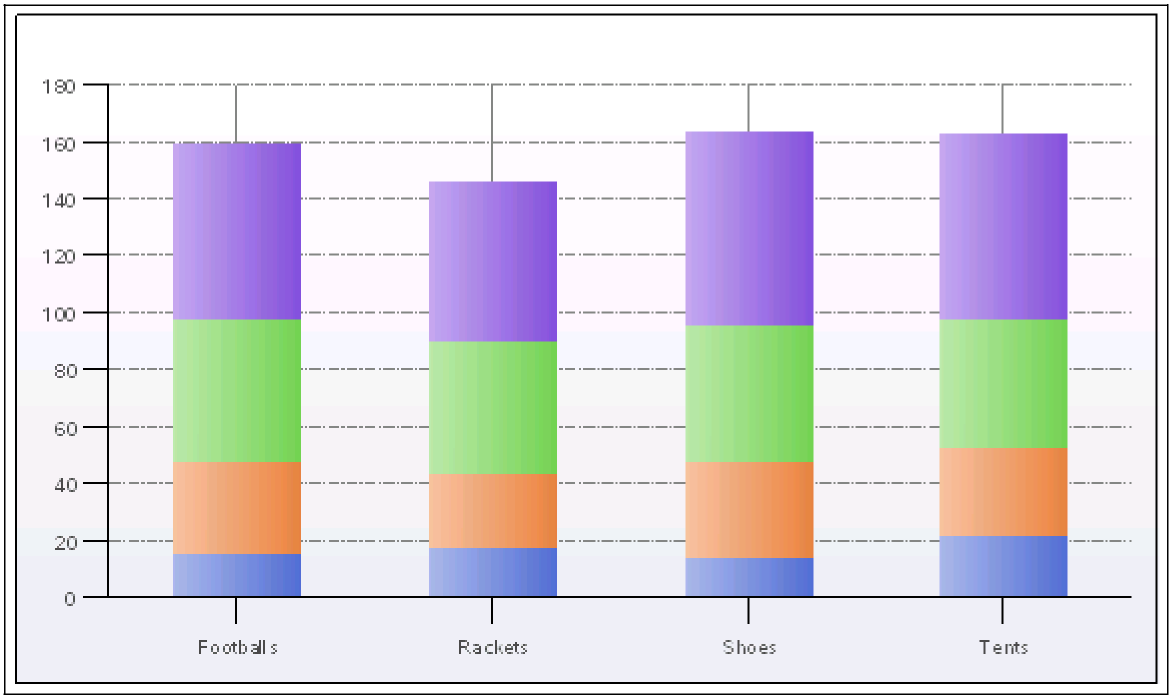
Horizontal bar chart

Stacked Bar Chart

With a the stacked bar chart, you can place a different emphasis on the same data. The stacked column shows total sales per region as well as the product segments that contribute to the total.

Use this line of code to specify a stacked bar chart:

```
&cChart.Type = %ChartType_2DStackedBar;
```

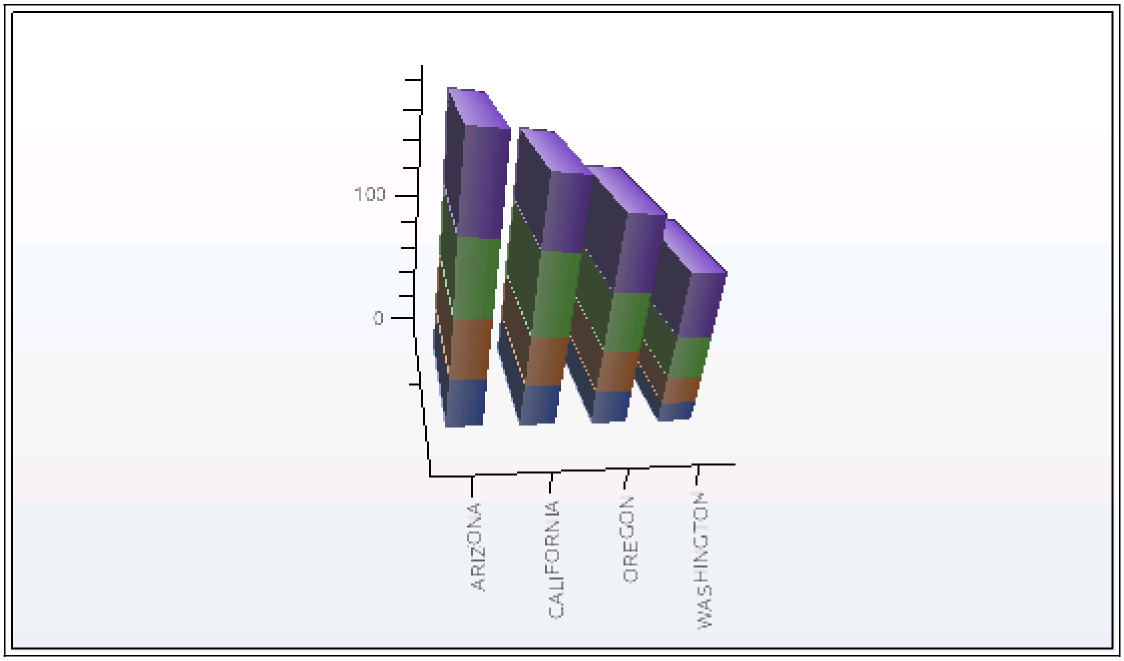



Stacked bar chart

3D Stacked Bar Chart

Use this line of code to specify a 3D stacked bar chart:

```
&cChart.Type = %ChartType_3DStackedBar;
```

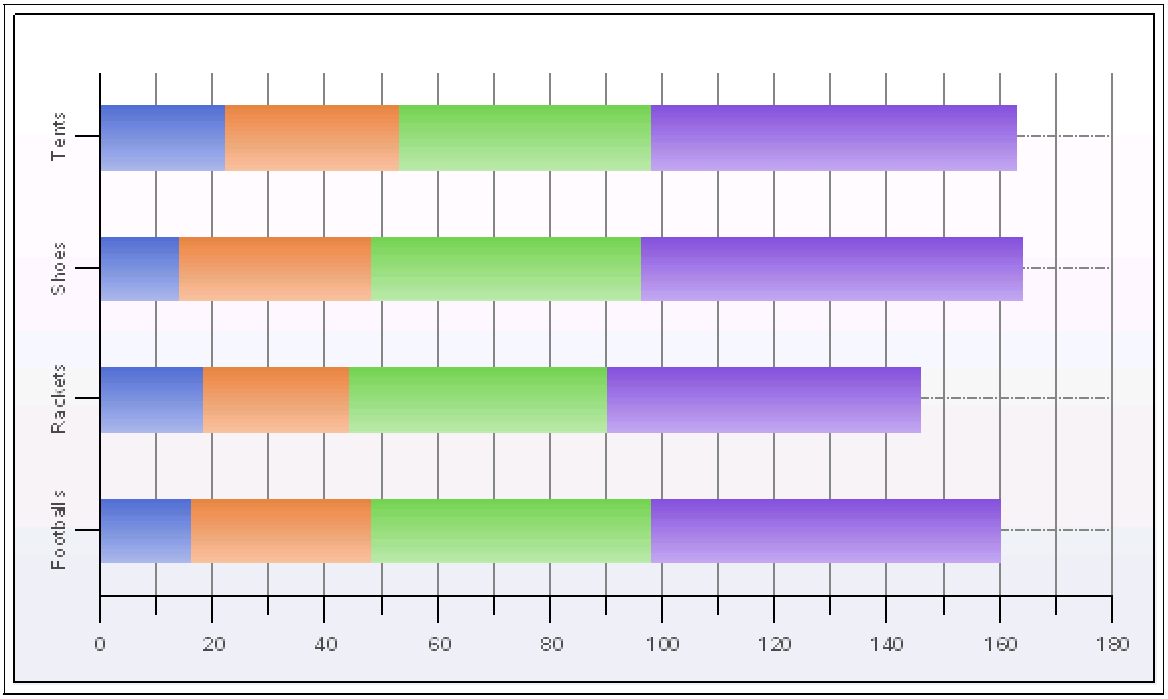


3D stacked bar chart

Horizontal Stacked Bar Chart

Use this line of code to specify a horizontal stacked bar chart:

```
&cChart.Type = %ChartType_2DHorizStackedBar;
```



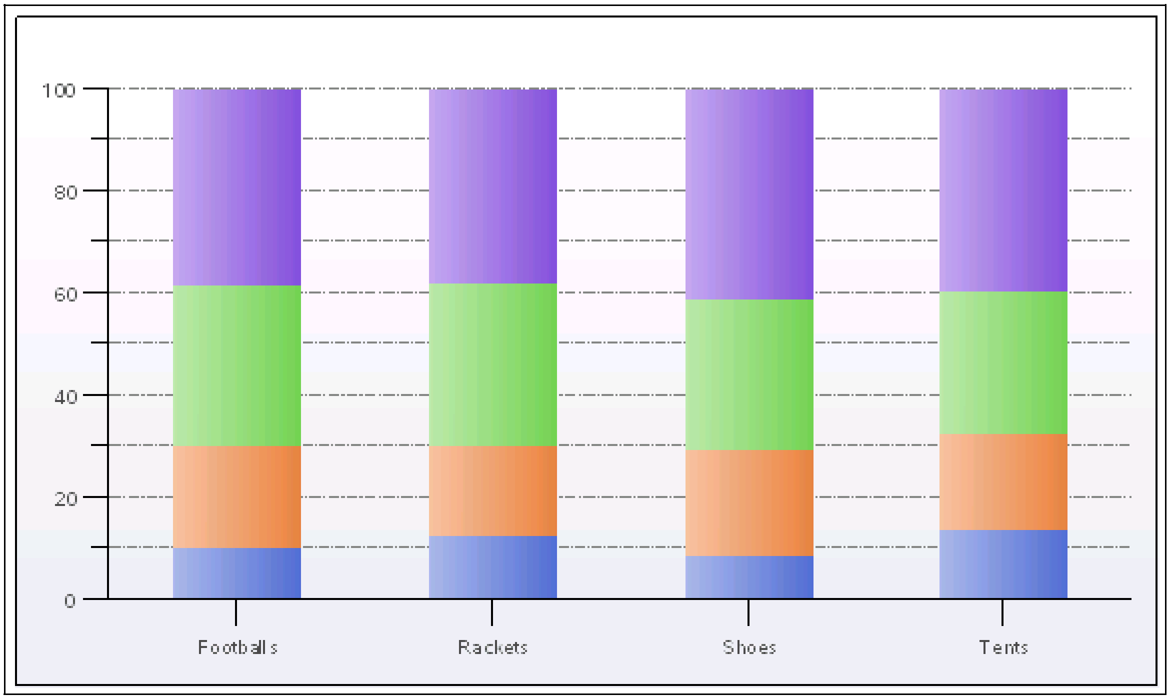
Horizontal stacked bar chart

Percent Bar Chart

A percent bar is similar to a stacked bar, but the segments of each bar add up to 100. The segments show the proportional contribution of each product to total sales.

Use this line of code to specify a horizontal stacked bar chart:

```
&cChart.Type = %ChartType_2DPercentBar;
```

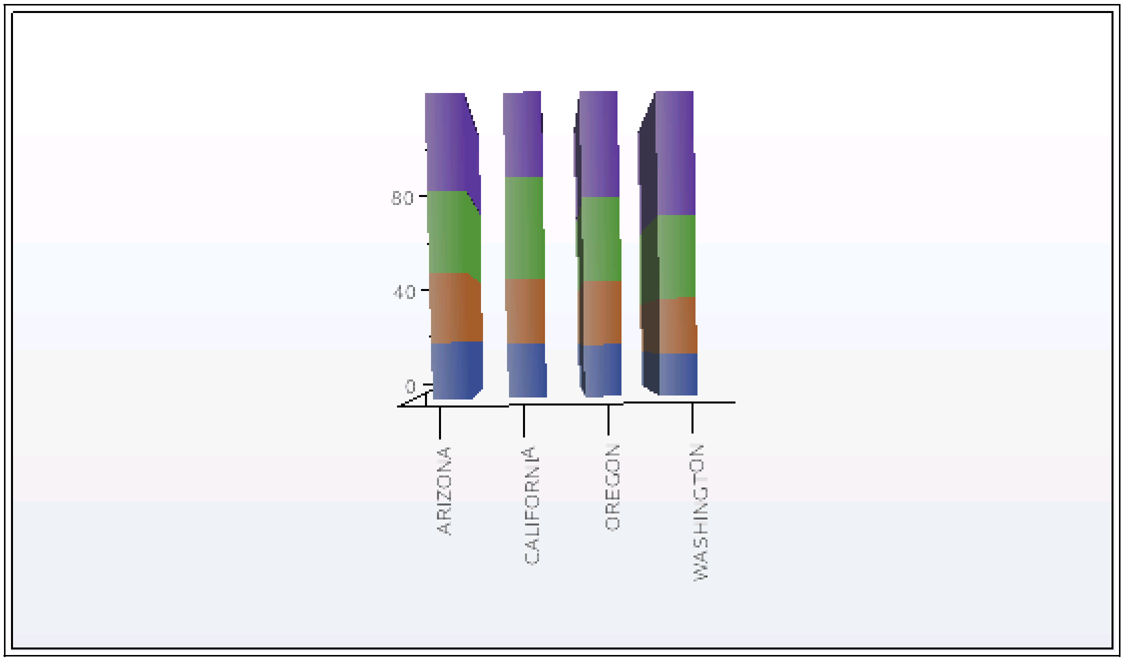



Percent bar chart

3D Percent Bar Chart

Use this line of code to specify a horizontal stacked bar chart:

```
&cChart.Type = %ChartType_3DPercentBar;
```

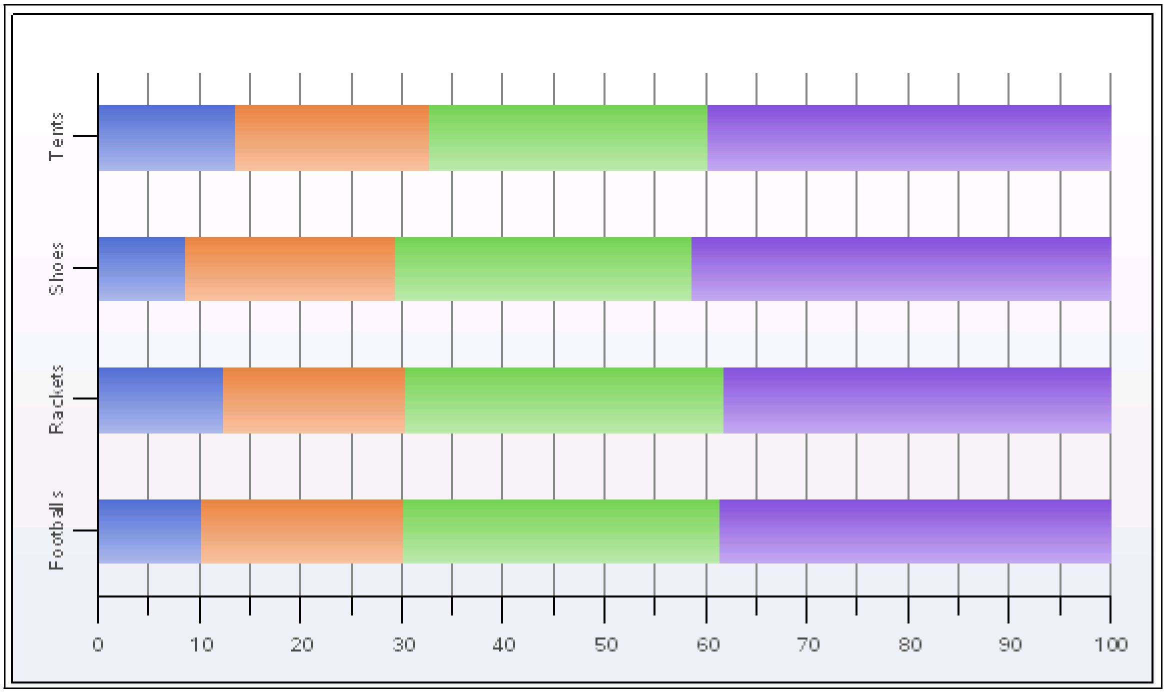


3D percent bar chart

Horizontal Percent Bar Chart

Use this line of code to specify a horizontal stacked bar chart:

```
&cChart.Type = %ChartType_2DHorizPercentBar;
```



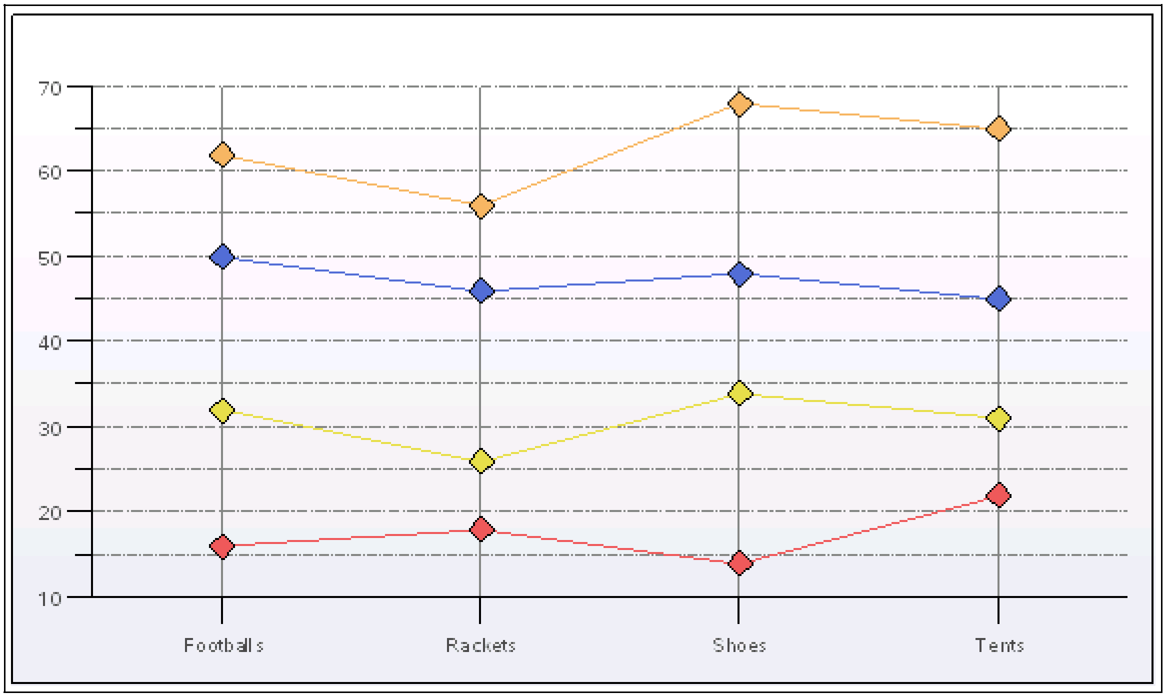
Horizontal percent bar chart

Line Chart

A line chart is useful for showing changes and trends over time or across categories.

Use this line of code to specify a horizontal stacked bar chart:

```
&cChart.Type = %ChartType_2DLine;
```

Line Chart

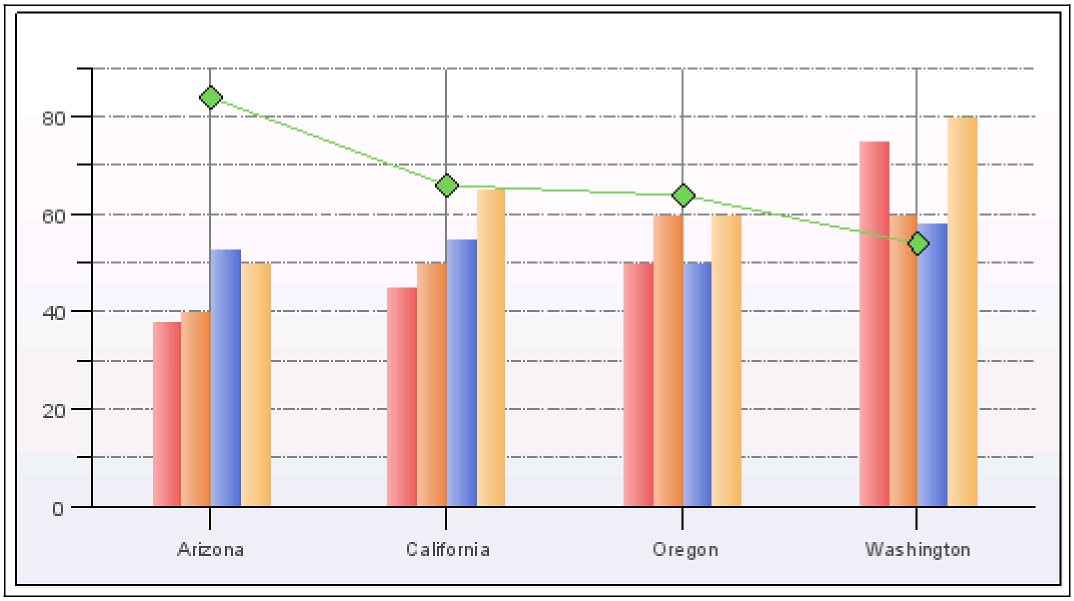
Overlay Chart

You can overlay one chart on top of another to compare two different but related sets of data. The following chart uses a second set of data, showing average temperature for each region.

Use the OLType property to specify whether the overlay is a line chart or a histogram.

Add this PeopleCode to the PeopleCode for your bar chart to create 2D line chart overlay:

```
/* Create OL chart */
&cChart.SetOLData(Record.SC_OLCHART_REC);
&cChart.SetOLDataSeries(SC_OLCHART_REC.SC_CHART_SERIES);
&cChart.SetOLDataXAxis(SC_OLCHART_REC.SC_CHART_XAXISDATA);
&cChart.SetOLDataYAxis(SC_OLCHART_REC.SC_YAXISDATA);
&cChart.OLType = %ChartType_2DLine;
```

Overlay chart

True XY Line Chart

Use the IsTrueXY property to create a true XY line chart.

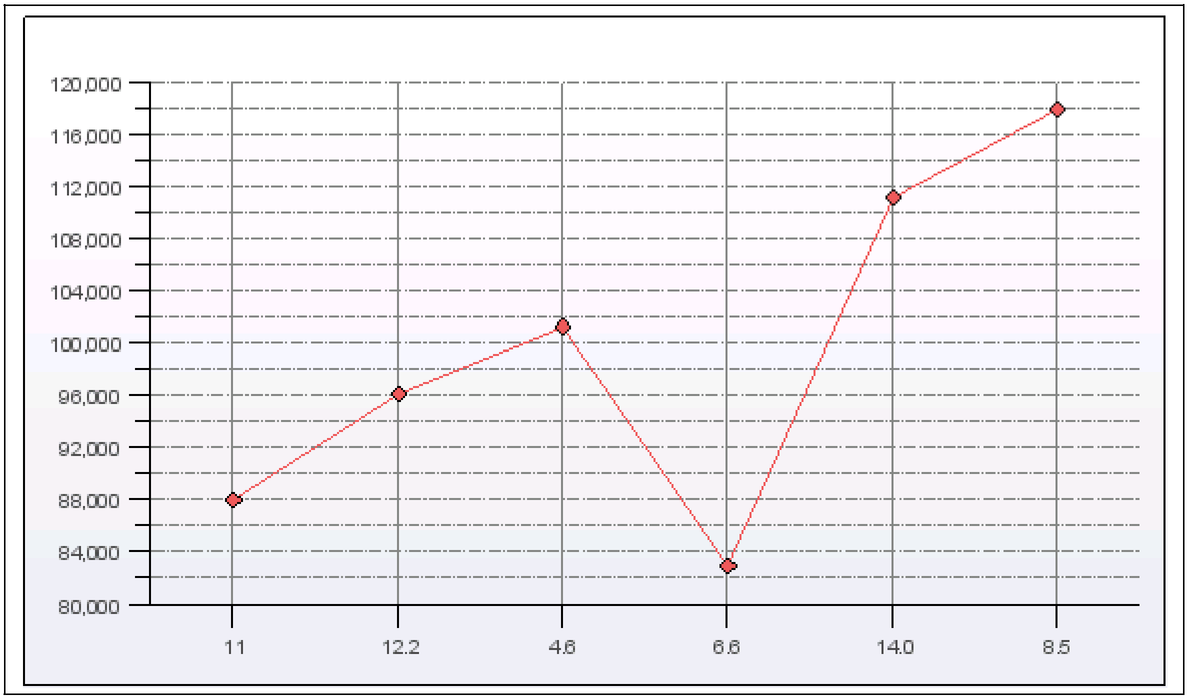
A true XY chart has a numeric X axis. That is, the numeric XY pairs can be plotted correctly.

By default, IsTrueXY is False. A default line chart plots X-axis values as discrete labels, ordered by record position.

For example, you have the following data:

X Axis	***Y Axis***
11	88000
12.2	96200
4.6	101300
6.6	83000
14	111200
8.5	11800

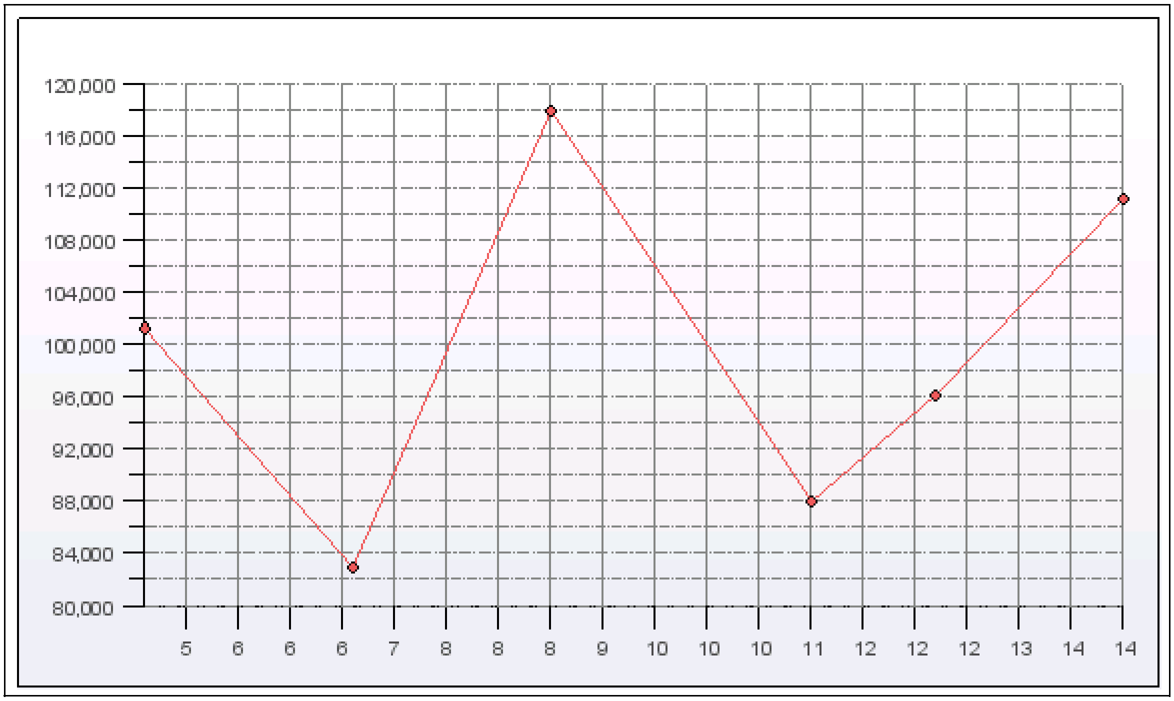
The following example shows a default line chart using the previous data:



Line chart with IsTrueXY = False

A line chart with IsTrueXY = True orders the X-axis values numerically on a continuous scale.

The following chart is a true XY line chart plotted with the same data.



Line chart with IsTrueXY = True

Scatter Chart

Use this line of code to specify a scatter chart:

```
&cChart.Type = %ChartType_2DHorizStackedBar;
```

A scatter chart compares number pairs. Each pair is plotted against the horizontal X axis and the vertical Y axis. The data values are scattered across the chart. XY scatter charts are good for showing comparisons of numbers, such as scientific or statistical data, where several measurements need to be plotted on a single chart.

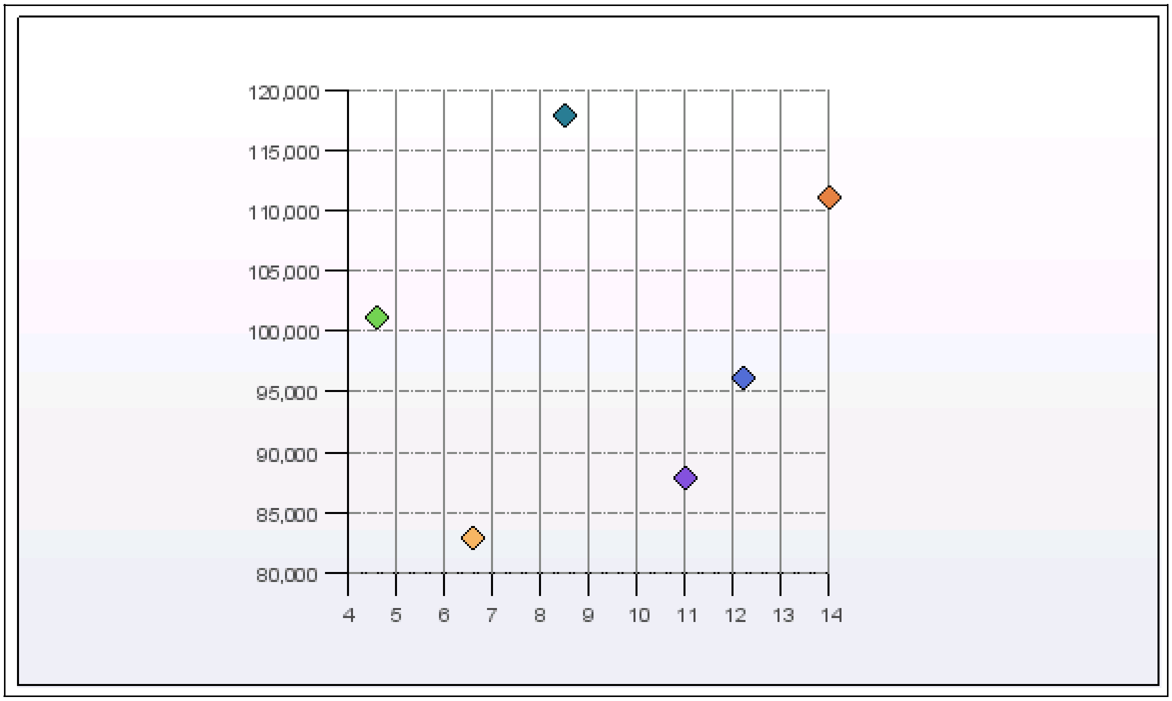
A scatter chart is a true XY chart, meaning that the X axis is numeric. The IsTrueXY property is automatically set to True.

For example, this sample data could be used to make an XY scatter chart:

Tenure	***Salary***
11	88000
12.2	96200
4.6	101300
6.6	83000
14	111200
8.5	11800

You use this code snippet to make a scatter chart:

```
&cChart.SetDataXAxis(SC_CHART_RECORD.SC_REGION);
&cChart.SetDataYAxis(SC_CHART_RECORD.SC_SALES);
&cChart.Type = %ChartType_2DScatter;
```

Scatter chart

You can set the color for each point, or glyph, on a scatter chart, and you can make each point drillable.

Note. If you have only one point of data, that is, only one X and Y pair, use a scatter chart. You may have unexpected results if you use a line chart with only one point.

Bubble Chart

A bubble chart is a special type of scatter chart.

If you add a third data set, glyph size, to a scatter chart, you can create a bubble chart, which uses the size of the markers to represent a third dimension of numeric data.

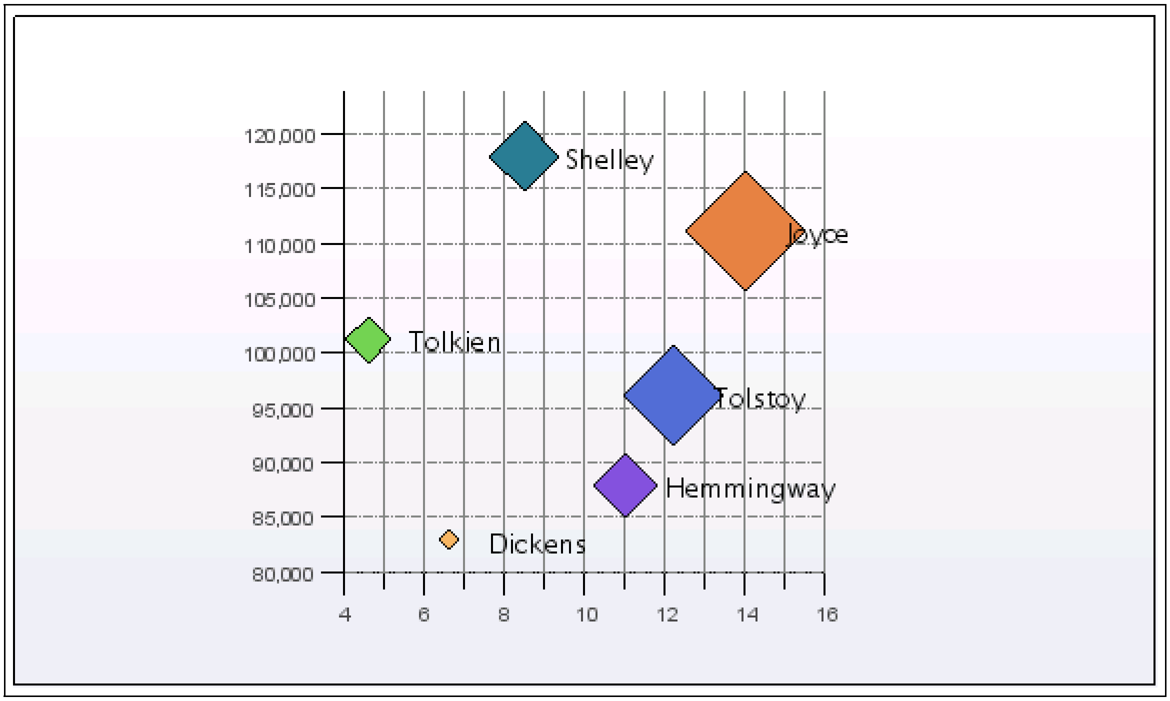
You can also set annotations for scatter charts and bubble charts.

This chart example uses Age to determine the size of the bubble markers, or glyphs, on the chart and Name to set the annotations:

Tenure	*Salary*	*Age*	*Name*
11	88000	39	Dickens
12.2	96200	55	Tolstoy
4.6	101300	32	Hemingway
6.6	83000	48	Tolkien
14	111200	59	Joyce

Tenure	Salary	Age	Name
8.5	11800	49	Shelley

```
&cChart.SetDataAnnotations(SC_CHART_RECORD.SC_NAME);
&cChart.SetDataGlyphScale(SC_CHART_RECORD.SC_AGE);
&cChart.Type = %ChartType_2DScatter;
```



Bubble chart

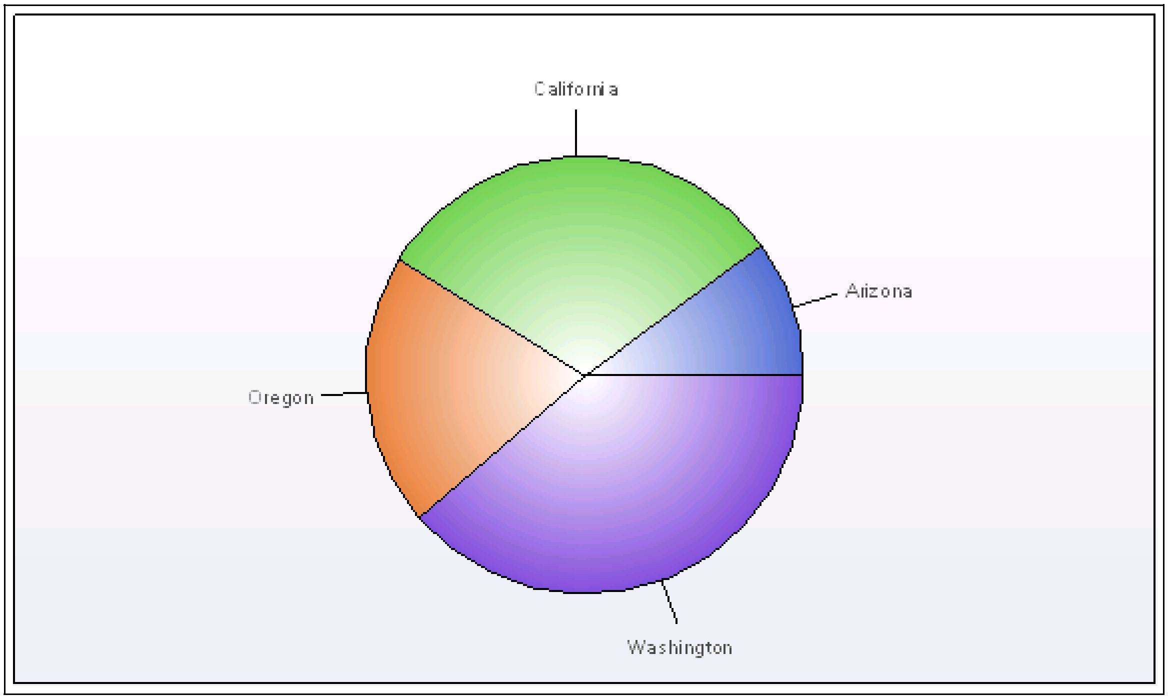
Pie Chart

A pie chart shows comparisons within a single set of values, and it shows how parts contribute to a whole. It is an ideal chart type to display the contribution of each region to an annual sales total.

A pie chart ignores the series and only plots X-axis data against Y-axis values. Our original bar chart has four regions times four products, resulting in 16 XY pairs. A pie chart with all of those slices would be confusing and not helpful. Instead, suppose you want to show just the data for tents, which are in rows 13–16. You can use a combination of the DataWidth and DataStartRow properties to select which rows to display.

You can make a pie chart showing only rows 13–16 by adding this snippet of code:

```
&cChart.DataWidth = 4;
&cChart.DataStartRow = 13;
&cChart.Type = %ChartType_2DPie;
```

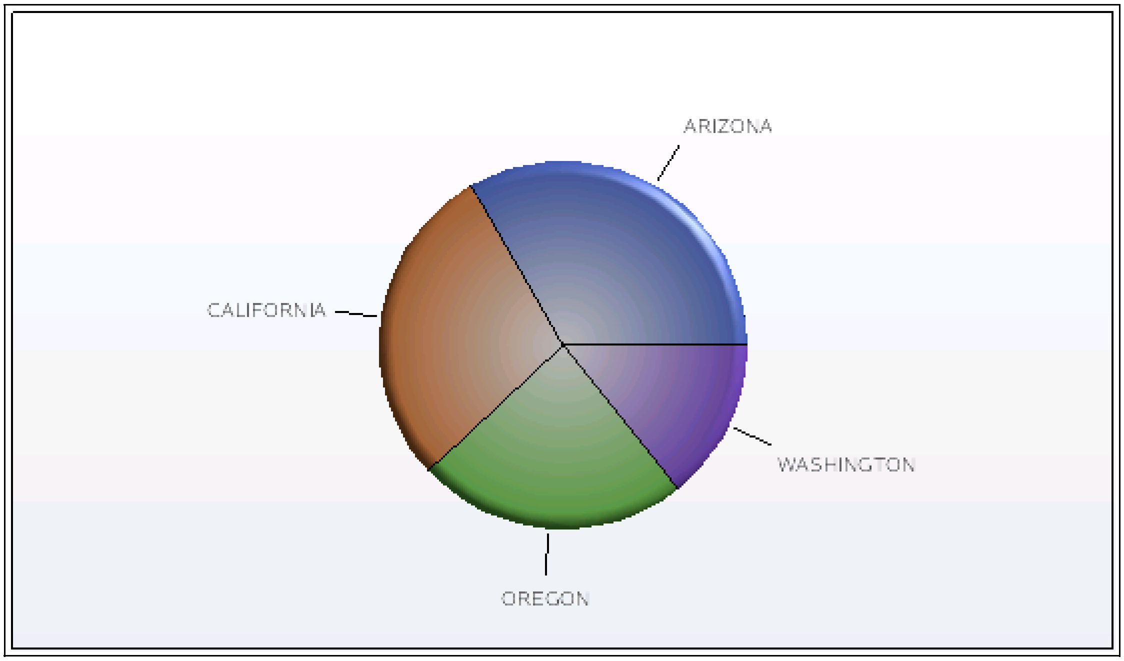


2D pie chart

3D Pie Chart

Use this line of code to specify a horizontal stacked bar chart:

```
&cChart.Type = %ChartType_3DPie;
```



3D pie chart

Tick Mark Considerations

If you want to remove the tick marks from the X or Y axis, you need to specify an empty array for the SetXAxisLabels or SetYAxisLabels methods.

The following example removes the tick marks from the Y axis:

```
&Arr = CreateArray("");
&Chart.SetYAxisLabel(&Arr);
```

If you set the X-axis labels to an empty array to remove the ticks, you must be plotting a single series on the chart because labels are automatically generated for each series if you do not provide them.

See Chapter 11, "Charting Classes," SetXAxisLabels, page 452 and Chapter 11, "Charting Classes," SetYAxisLabels, page 452.

Adding Drilldown

If you set the property IsDrillable to True and add FieldChange PeopleCode to your chart, your user can get more information by clicking a chart data element—a bar, line segment, data point, or pie slice.

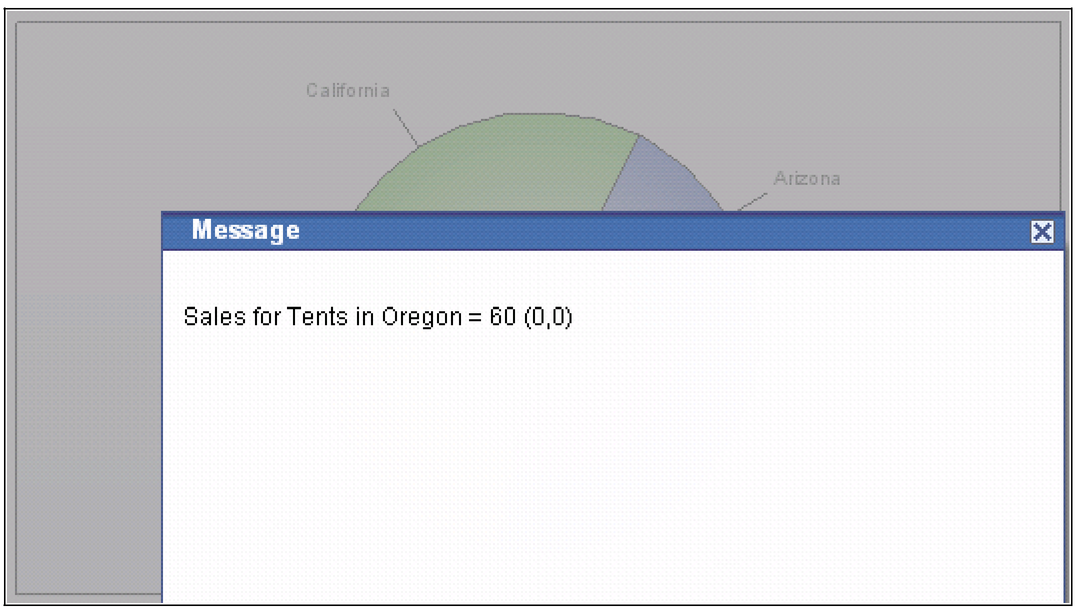
For a simple example, add this PeopleCode to the FieldChange event on the Y-axis field:

```
MessageBox(0, "", 0, 0, " Sales for " | SC_CHART_RECORD.SC_PRODUCT | " in "
  | SC_CHART_RECORD.SC_REGION | " = " | SC_CHART_RECORD.SC_SALES);
```

And add this code to the page Activate event for your pie chart:

```
&cChart.IsDrillable = True;
```

When the user clicks the Oregon slice of the pie chart, this message appears:



Example of drilldown triggering PeopleCode with MessageBox

Current context provides the values for fields in the current row in the component buffer. You can use drilldown with your chart data fields along with any other data in the component buffer to accomplish anything FieldChange PeopleCode is capable of.

If the bars on a bar chart are too narrow, a user may not be able to click them. Sometimes, a few bars on a chart are affected, sometimes all of the bars cannot be clicked. The workaround is to put fewer bars on your chart or to make your chart bigger so that each bar is wider.

See Chapter 11, "Charting Classes," IsDrillable, page 457.

Using the Other Charting Classes

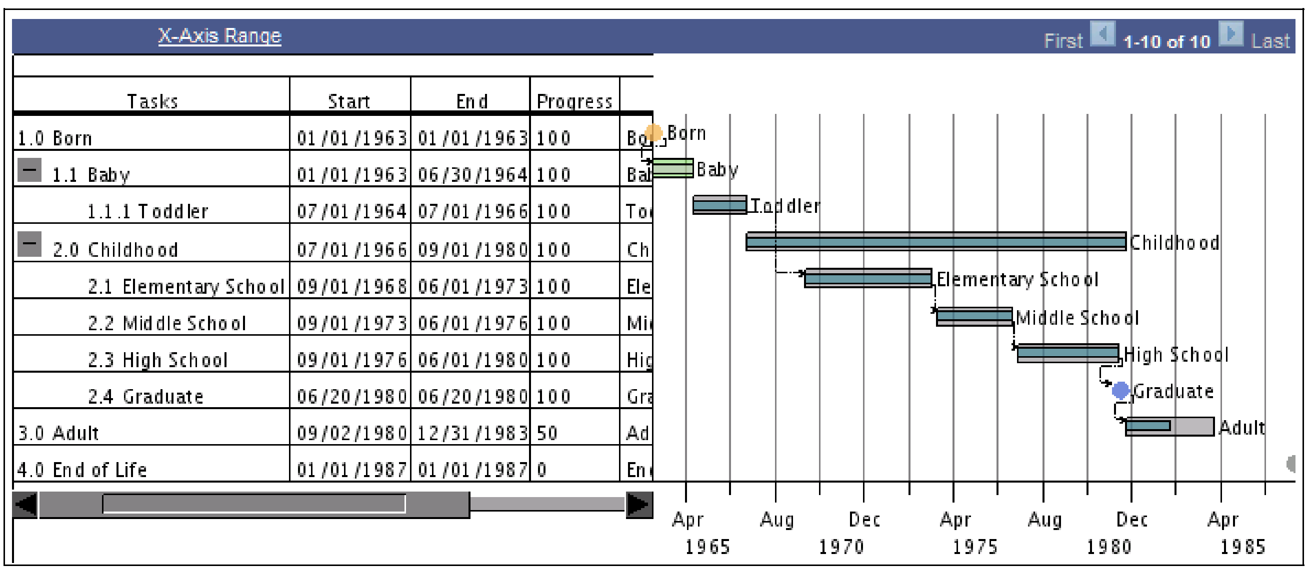
In addition to the all the charts you can create with the Chart class, you can create specialized charts using the Gantt class, the OrgChart class, and the RatingBoxChart class.

You follow the same initial steps to create these specialized charts as you do for the Chart class chart types. The PeopleCode programs to set the data and properties are similar as well, but Gantt charts, organization charts, and rating box charts use data differently and interact with the user in different ways than do the Chart class charts.

See Chapter 11, "Charting Classes," Creating a Gantt Chart, page 585; Chapter 11, "Charting Classes," Creating an Organization Chart, page 588 and Chapter 11, "Charting Classes," Creating a Rating Box Chart, page 592.

Creating a Gantt Chart

The PeopleCode and data used in this example create the following Gantt chart:



Example of a Gantt chart

This example uses the following data in the record QE_GANTT_TASK:

Customize | Find | | First 1-10 of 10 Last

	*Task ID	TASK NAME	*PLANNED START DATE	*PLANNED END DATE
1	1	Born	01/01/1963 12:00AM	01/01/63 12:00AM
2	2	Baby	01/01/1963 12:00AM	06/30/64 12:00AM
3	3	Toddler	07/01/1964 12:00AM	07/01/66 12:00AM
4	4	Childhood	07/01/1966 12:00AM	09/01/80 12:00AM
5	5	Elementary School	09/01/1968 12:00AM	06/01/73 12:00AM
6	6	Middle School	09/01/1973 12:00AM	06/01/76 12:00AM
7	7	High School	09/01/1976 12:00AM	06/01/80 12:00AM
8	8	Graduate	06/20/1980 12:00AM	06/20/80 12:00AM
9	9	Adult	09/02/1980 12:00AM	12/31/83 12:00AM
10	10	End of Life	01/01/1987 12:00AM	01/01/87 12:00AM

Example of Gantt task data

The chart also uses the following dependency data in the record QE_GANTT_TASKD:

Task ID	Parent Task ID	URL
2	1	http://www.oracle.com
5	3	http://www.oracle.com
6	5	http://www.oracle.com
7	6	http://www.oracle.com
8	7	http://www.oracle.com
9	8	http://www.oracle.com

Example of Gantt dependency data

To create a Gantt chart:

1. In Application Designer, add a chart control to a page and associate the chart control with a record name and a field name.

   See Chapter 11, "Charting Classes," Creating a Chart Using the Chart Class, page 557.

2. Initialize a Gantt chart object.

   Initialize the chart using the GetGanttChart function. The arguments of the function are the record and field name specified for the chart definition in Application Designer.

```
Local Gantt &gGantt;
&gGantt = GetGanttChart(QE_CHART_DUMREC.QE_CHART_FIELD);
```


3. Specify values for the required methods.

   At a minimum, specify the source of the task data and which fields contain task ID, start date, and end date.

```
&gGantt.SetTaskData(Record.QE_GANTT_TASK);
&gGantt.SetTaskID(QE_GANTT_TASK.QE_TASK_ID);
&gGantt.SetPlannedStartDate(QE_GANTT_TASK.QE_PLANNED_START);
&gGantt.SetPlannedEndDate(QE_GANTT_TASK.QE_PLANNED_END);
```

4. Specify values for the additional methods controlling the appearance of the data.

   After setting the required methods, you can set additional methods to control the appearance of the chart. The following code specifies the amount that the scroll bars scroll when clicked, the percentage of the entire chart that is taken up by the chart area, the width and height of the chart, on what row the chart should begin displaying data, the width of the data, if the user can click the chart area of the Gantt chart and either execute a function or a URL, and whether labels should appear with the tasks in the task section.

```
&gGantt.SetTableXScrollbar(0.20);
&gGantt.SetChartArea(50);
&gGantt.Width = 700;
&gGantt.Height = 380;
&gGantt.DataStartRow = 1;
&gGantt.DataWidth = 11;
&gGantt.IsDrillable = True;
&gGantt.ShowTaskLabels = True;
```

5. Specify the appearance of grid lines.

   Depending on the type of data that you are using, you can specify the type of grid lines that should appear in the chart section of the Gantt chart.

```
&gGantt.GridLineType = %ChartLine_Solid;
&gGantt.GridLines = %ChartGrid_Vertical;
```

6. Specify the appearance of data.

   The following methods control the appearance of the different parts of the task data, such as the label, which hints appear when you pass the mouse over the task data, and so on.

```
&gGantt.SetTaskName(QE_GANTT_TASK.QE_TASK_NAME);
&gGantt.SetWBSNumbering(QE_GANTT_TASK.QE_HINTS);
&gGantt.SetTaskLevel(QE_GANTT_TASK.QE_LEVEL);
&gGantt.SetTaskLabel(QE_GANTT_TASK.QE_TASK_LABEL);
&gGantt.SetTaskProgress(QE_GANTT_TASK.QE_TASK_PROGRESS);
&gGantt.SetTaskHints(QE_GANTT_TASK.QE_HINTS);
&gGantt.SetTaskBarURL(QE_GANTT_TASK.QE_URL);
&gGantt.SetTaskMilestone(QE_GANTT_TASK.QE_TASK_MILESTONE);
&gGantt.TaskMilestoneGlyph = QE_GANTT_PROJ.QE_MILESTONE_GLYPH;
&gGantt.SetTaskExpanded(QE_GANTT_TASK.QE_EXPANDED);

&gGantt.SetPlannedTaskBarColor(QE_GANTT_TASK.QE_PLANNED_COLOR);
&gGantt.SetActualTaskBarColor(QE_GANTT_TASK.QE_ACTUAL_COLOR);
&gGantt.SetTaskProgressBarColor(QE_GANTT_TASK.QE_PROGRESS_COLOR);
```

7. Specify the additional fields that will appear in the table section.

   You can also specify application data that will appear with the task data.

```
&gGantt.SetTaskAppData(QE_GANTT_TASK.QE_PLANNED_START,
   QE_GANTT_TASK.QE_PLANNED_END, QE_GANTT_TASK.QE_TASK_PROGRESS,
   QE_GANTT_TASK.QE_TASK_NAME, QE_GANTT_TASK.QE_EXPANDED);
```

8. Specify titles for the task bars.

```
&gGantt.TaskTitle = "Tasks";
 &appDataArray = CreateArray("Start", "End", "Progress", "Name", "Expanded");
 &gGantt.SetTaskAppDataTitles(&appDataArray);
```

9. Specify the fields for task dependency data.

   You do not need to specify dependency data. However, if you do, you must also specify the parent ID and the child ID (where the dependency data starts and ends). You can also specify the type of line used between dependencies and a URL that the browser will open to if the user clicks a dependency line.

```
&gGantt.SetTaskDepenencyData(Record.QE_GANTT_TASKD);
&gGantt.SetTaskDependencyParentID(QE_GANTT_TASKD.QE_PARENT_TASK_ID);
&gGantt.SetTaskDependencyChildID(QE_GANTT_TASKD.QE_TASK_ID);
&gGantt.TaskDependencyLineType = %ChartLine_Solid;
&gGantt.SetTaskDependencyURL(QE_GANTT_TASKD.URL);
```

10. Specify the actual dates for the project.

    If you know the actual start and end dates, in addition to the planned date, you can add those to your chart.

```
&gGantt.SetActualStartDate(QE_GANTT_TASK.QE_ACTUAL_START);
&gGantt.SetActualEndDate(QE_GANTT_TASK.QE_ACTUAL_END);
```

11. Specify the DateTime axis values.

    You can specify values for where the DateTime axis starts and ends.

```
&gGantt.AxisStartDateTime = DateTimeValue("01/01/1963 8:00:00");
&gGantt.AxisEndDateTime = DateTimeValue("01/01/1987 8:00:00");
```

12. Specify the format of the DateTime axis values.

    You can specify the format of the DateTime elements, such as the appearance of the days of the week, whether as a number or using the full name, and so on.

```
&gGantt.SetMonthFormat(%Chart_MonthFormat_FullName);
&gGantt.SetDayFormat(%Chart_DayFormat_2Digit);
```

Creating an Organization Chart

The PeopleCode used in this example creates the following organization chart:

Example organization chart

To create an organization chart:

1. Add a chart control to a page in PeopleSoft Application Designer.

   Associate the chart control with a record name and field name.

   See Chapter 11, "Charting Classes," Creating a Chart Using the Chart Class, page 557.

2. Add the organization node record and the pop-up node record to the component.

   These records are derived/work records that contain clones of the subrecords PTORGNODE_SBR and PTPOPUPNODE_SBR. These records must be in the component buffer. You can add them to a hidden grid on the page, or you can place them on another page in the same component.

3. If the chart will have breadcrumbs, add the breadcrumb node record to the component.

   This derived/work record contains a clone of the subrecord PTORGCRMB_SBR. The record must be in the component buffer.

4. Prepare the organization chart data.

   In an event such as PreBuild, instantiate two rowset objects and populate them with the organization node data and the pop-up node data.

   This example shows one possible way. How you do it depends on how your data is stored and how you intend to present it.

```
Component Rowset &rs, &rsP, &rsBC, &rsOrgNode, &rsOrgPopupND, &rsBreadCrumbs;
Component OrgChart &ocOrgChart;

Local Rowset &RSGridDR;
Local Rowset &RSAGRID;

/* The record QE_ORG_NODEDATA has the data for the org chart **
** You will probably fill the rowset with some subset of the data**
** in the database. This example uses all the rows. */
&rs = CreateRowset(Record.QE_ORG_NODEDATA);
&rs.Fill();

/* Get the grid on the page for the node data and initalize to nulls */
& rsOrgNode = GetLevel0()(1).GetRowset(Scroll.QE_ORG_NODE);
& rsOrgNode.Flush();

/* Copy data from database table, QE_ORG_NODEDATA to the derived/work
** record QE_ORG_NODE_DR */
&rs.CopyTo(&rsOrgNode, Record.QE_ORG_NODEDATA, Record.QE_ORG_NODE_DR);

/* The record QE_ORG_POPDATA has the data for the popup chart */
&rsP = CreateRowset(Record.QE_ORG_POPDATA);
&rsP.Fill();

/* Get the grid on the page for the popup data and initalize to nulls */
& rsOrgPopupND = GetLevel0()(1).GetRowset(Scroll.QE_ORG_POPUP);
& rsOrgPopupND.Flush();

/* Copy data from database table, QE_ORG_POPDATA to the derived/work
** record QE_ORG_POP_DR */
&rsP.CopyTo(&rsOrgPopupND, Record.QE_ORG_POPDATA, Record.QE_ORG_POP_DR);

/* Fill in the breadcrumb information for this chart if the chart **
** uses breadcrumbs. */
&rsBC = CreateRowset(Record.QE_ORG_CRMB);
&rsBC.Fill();

/* Get the grid on the page for the pop data and initalize to nulls */
&rsBreadCrumbs = GetLevel0()(1).GetRowset(Scroll.QE_ORG_BCRMB);
&rsBreadCrumbs.Flush();

/* Copy data from database table, QE_ORG_CRMB to the derived/work
** record QE_ORG_CRMB_DR */
&rsBC.CopyTo(&rsBreadCrumbs, Record.QE_ORG_CRMB, Record.QE_ORG_CRMB_DR);
```

5. Instantiate an OrgChart object.

   In an event such as Activate, add PeopleCode to instantiate and define your organization chart.

   Get the chart using the GetOrgChart function. The argument for this function is the record name and field name specified on the Records tab of the Chart Properties dialog box in Application Designer.

```
Component Rowset & rsOrgNode, & rsOrgPopupND, &rsBreadCrumbs;
Component OrgChart &ocOrgChart;
&ocOrgChart= GetOrgChart(QE_CHART_DUMREC.QE_CHART_FIELD);
```


6. Specify the organization node record and the pop-up node record.

   These records contain the clones of the subrecords PTORGNODE_SBR and PTORGPOPUP_SBR, respectively.

```
&ocOrgChart.SetNodeRecord (Record.QE_ORG_NODE_DR);
&ocOrgChart.SetPopUpNodeRecord (Record.QE_ORG_POP_DR);
```

7. Specify the node data and pop-up node data rowset objects.

```
/*&RSORGND and &RSORGPOPUPND are level 1 component rowsets that contain the
** organization node data and pop-up node data respectively.*/

&ocOrgChart.SetNodeData (&rsOrgNode);
&ocOrgChart.SetPopUpNodeData (&RSORGPOPUPND);
```

8. (Optional) Specify values for the chart display properties.

```
&ocOrgChart.MainTitle = "Application Development";
&ocOrgChart.MainTitleStyle = "PT_ORGCHART_TITLE";
&ocOrgChart.Direction = 2;
&ocOrgChart.Height = 380;
&ocOrgChart.Width = 700;
&ocOrgChart.VerticalSpace = 15;
&ocOrgChart.ImageLocation = 1;
&ocOrgChart.Style= "PTORGCHART";
&ocOrgChart.HasLegend = True;
&ocOrgChart.NodeMaxDisplayDescLength =30;
&ocOrgChart.CollapsedImage ="PT_COLLAPSED_CHART";
&ocOrgChart.ExpandedImage ="PT_EXPANDED_CHART";
&Expanded_Msg=MsgGetText(110, 100);
&Collapsed_Msg=MsgGetText(110, 101);
&ocOrgChart.Expanded_Msg =&Expanded_Msg
&ocOrgChart.Collapsed_Msg =&Collapsed_Msg
```

9. Specify values for the legend. This is required only if the chart has a legend.

```
&ocOrgChart.LegendStyle = PT_ORGCHART_LEGEND;
&LegendArray = CreateArray("Main", "1-2yrs", "3-5yrs", "Emergency",
"Retirement", "Key", "Successors");
&LegendImgArray = CreateArray("QE_NOWORGCHART", "QE_12ORGCHART",
"QE_34ORGCHART", "QE_EMRORGCHART", "QE_RETIRORG", "QE_KPERSON",
"QE_SUCCSORGCHART");

&ocOrgChart.SetLegend(&LegendArray);
&ocOrgChart.SetLegendImg(&LegendImgArray);
&ocOrgChart.LegendPosition = %ChartLegend_Top;
&ocOrgChart.LegendStyle = "PT_ORGCHART_LEGEND";
```

10. (Optional) Specify image values.

```
&ocOrgChart.ImageLocation =1;
&ocOrgChart.ImageHeight =150;
&ocOrgChart.ImageMouseoverMagnificationFactor =150;
&ocOrgChart.DefaultImage=PT_CHART_DEFAULTIMG;
```

11. Specify the breadcrumb data and the rowset and display properties. These are required only if your chart uses breadcrumbs.

```
&ocOrgChart.SetCrumbRecord(Record.QE_ORG_CRMB_DR);
&ocOrgChart.SetCrumbData(&rsBreadCrumbs);
&ocOrgChart.CrumbMaxDisplayLength = 24;
&ocOrgChart.CrumbSeparatorImage = "PT_ORG_BRCRM_SEP";
&ocOrgChart.CrumbDescrStyle = "PT_ORGCHART_BRDCRM";
```

12. (Optional) Specify the maximum number of nodes that will appear in the pop-up without a vertical scroll bar.

```
&ocOrgChart.MaxPopUpDisplayNode = 2;
```

13. (Optional) Assign style class names to control the styles of each node descriptor.

    If you do not specify a style class name, then the PeopleTools default style class is used.

    This example uses the default style class names, so this segment of code could be omitted. It is shown for demonstration purposes only.

```
&ocOrgChart.NodeDescr1Style = "PT_ORGNODE_DESC1";
&ocOrgChart.NodeDescr2Style = "PT_ORGNODE_DESC2";
&ocOrgChart.NodeDescr3Style = "PT_ORGNODE_DESC3";
&ocOrgChart.NodeDescr4Style = "PT_ORGNODE_DESC4";
&ocOrgChart.NodeDescr5Style = "PT_ORGNODE_DESC5";
&ocOrgChart.NodeDescr6Style = "PT_ORGNODE_DESC6";
&ocOrgChart.NodeDescr7Style = "PT_ORGNODE_DESC7";
```

14. (Optional) Assign style class names to control the styles of each pop-up node descriptor.

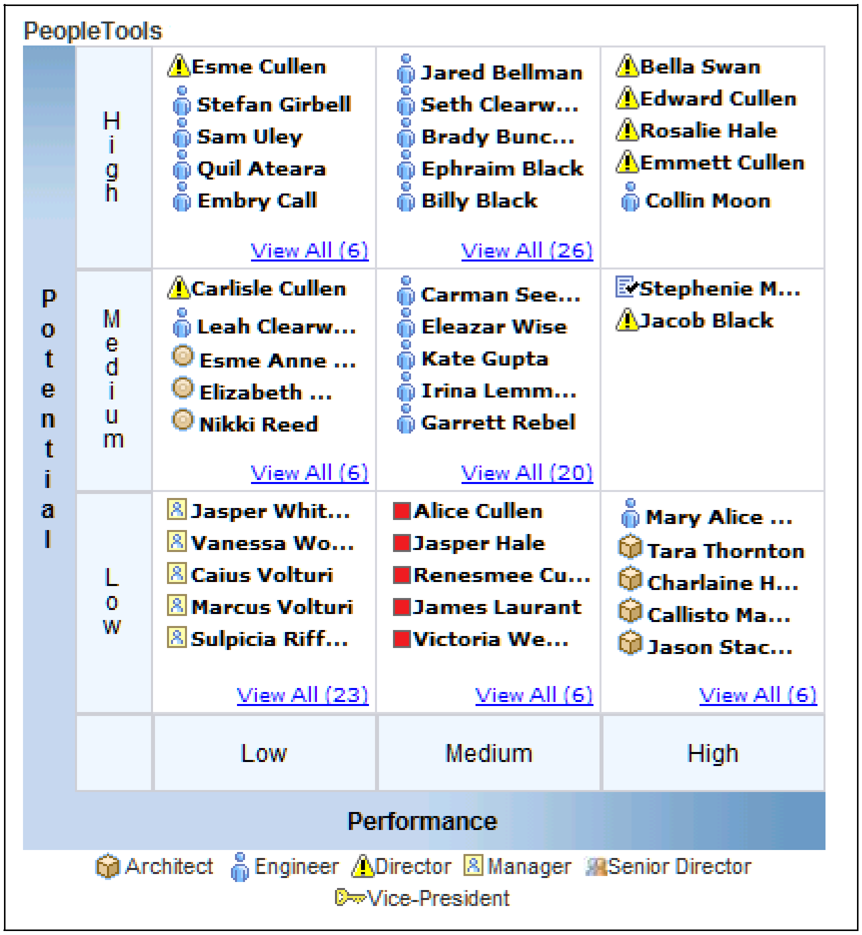
    If you do not specify a style class name, then the PeopleTools default style sheet is used.

    This example uses the default style class names, so this segment of code could be omitted. It is shown for demonstration purposes only.

```
&ocOrgChart.PopupNodeDescr1Style = "PT_POPNODE_DESC1";
&ocOrgChart.PopupNodeDescr2Style = "PT_POPNODE_DESC2";
&ocOrgChart.PopupNodeDescr3Style = "PT_POPNODE_DESC3";
&ocOrgChart.PopupNodeDescr4Style = "PT_POPNODE_DESC4";
&ocOrgChart.PopupNodeDescr5Style = "PT_POPNODE_DESC5";
&ocOrgChart.PopupNodeDescr6Style = "PT_POPNODE_DESC6";
&ocOrgChart.PopupNodeDescr7Style = "PT_POPNODE_DESC7";
&ocOrgChart.PopupNodeDescr8Style = "PT_POPNODE_DESC8";
&ocOrgChart.PopupHeaderStyle = "PT_POPNODE_HEADER";
```

Creating a Rating Box Chart

The PeopleCode used in the following example creates this rating box chart:



Example rating box chart

Follow these steps to create a rating box chart.

1. Create the rating box chart node record.

   a. Clone the PTRATINGBOX_SBR subrecord to create an application-specific subrecord.

   b. Create a new application-specific record and insert the new subrecord.

2. Add a chart control to a page in Application Designer.

   See Chapter 11, "Charting Classes," Creating a Chart Using the Chart Class, page 557.

3. Add the rating box chart node record to the component.

   This record must be in the component buffer. If you do not want to give the user access to the data you can add the record to a hidden grid on the page. Alternatively, you can place it on another page in the same component.

4. Add PeopleCode.

   - In an event such as PreBuild, add PeopleCode to create a component buffer rowset that references the rating box chart node record.
   - Add PeopleCode, probably in the page Activate event, to populate the rowset and define the chart.
   - Add FieldChange PeopleCode that will execute when drag-and-drop events occur.

PeopleCode for the Rating Box Chart Example

Complete these steps to create a rating box chart:

1. Prepare the data for the chart.

   Typically, write PeopleCode in an event such as PreBuild to populate a rowset with the data you want to display in the rating box chart.

```
Component Rowset &rs, &&rsRatingBox;
Component RatingBoxChart &rbRatingBoxChart;

/* the record QE_RATEBOX_DATA has the data for the RATING BOX  chart */
&rs = CreateRowset(Record.QE_RATEBOX_DATA);
&rs.Fill("Where QE_RATEBOX_TC=:1", QE_RATEBOX_TC.QE_RATEBOX_TC.Value);

/* Get the grid on the page for the node data and initalize to nulls */
&rsGrid = GetLevel0()(1).GetRowset(Scroll.QE_RATEBOX_TC);
&rsGrid.Flush();

/* Copy data from database table, QE_RATEBOX_DATA, to the
level 1 component rowset associated with the component derived working
record QE_RATEBOX_DR */

&rs.CopyTo(&rsGrid, Record.QE_RATEBOX_DATA, Record.QE_RATEBOX_DR);
```

2. Define the RatingBoxChart.

   Add PeopleCode in an event such as Activate.

   Instantiate a RatingBoxChart object using the GetRatingBoxChart built-in function to reference the page control field of the chart.

```
Component Rowset &rs, &rsGrid;
Component RatingBoxChart &rbRatingBoxChart;

&rbRatingBoxChart = GetRatingBoxChart(QE_CHART_DUMREC.QE_CHART_FIELD);
```

3. Set the required display properties for the chart.

```
&rbRatingBoxChart.XaxisBoxNum = 3;
&rbRatingBoxChart.YaxisBoxNum = 3;
&rbRatingBoxChart.PopUpWidth = 200;
&rbRatingBoxChart.PopUpHeight = 200;
&XAxisArray = CreateArray("Low", "Medium", "High");
&YAxisArray = CreateArray("Low", "Medium", "High");

&rbRatingBoxChart.SetXAxisLabels(&XAxisArray);
&rbRatingBoxChart.SetYAxisLabels(&YAxisArray);
```


4. (Optional) Specify other values for the chart display properties.

   If you do not specify these properties, the following default value will be used:

```
&rbRatingBoxChart.IsDragable = True;
&rbRatingBoxChart.BoxMaxDisplayItems = 3;
&rbRatingBoxChart.NDMaxDisplayDescLength = 25;
&rbRatingBoxChart.ShowNodeDescription = True;
&rbRatingBoxChart.GridLineType = %ChartLine_Solid;
&rbRatingBoxChart.XAxisTitle = "Performance";
&rbRatingBoxChart.YAxisTitle = "Potential";
&rbRatingBoxChart.NDMaxDisplayDescLength = 15;
&rbRatingBoxChart.BoxMaxDisplayItems = 5;

&rbRatingBoxChart.Height = 400;
&rbRatingBoxChart.Width = 400;
&rbRatingBoxChart.ShowNodeDescription = true;
&rbRatingBoxChart.GridLineType=%ChartLine_Solid;
&rbRatingBoxChart.IsDragable = True;
&rbRatingBoxChart. BoxMaxDisplayItems= 1;
&rbRatingBoxChart. NDMaxDisplayDescLength = 50;
```

5. (Optional) Assign style class names for the chart and X/Y Axis title and labels.

   If you do not specify a style class name, the PeopleTools default style sheet is used.

```
&rbRatingBoxChart.Style = "PSCHARTDEFAULT";
&rbRatingBoxChart.XAxisTitleStyle = "PT_RATBOX_XTTL";
&rbRatingBoxChart.YAxisTitleStyle = "PT_RATBOY_YTTL";
&rbRatingBoxChart.XAxisLabelStyle = "PT_RATBOX_XAXIS";
&rbRatingBoxChart.YAxisLabelStyle = "PT_RATBOX_YAXIS";
```

6. (Optional) Specify title and legend properties.

```
&rbRatingBoxChart.MainTitle = "Development Division";
&rbRatingBoxChart.MainTitleStyle="PT_RATBOX_TITLE"
&rbRatingBoxChart.HasLegend = True;
&rbRatingBoxChart.LegendPosition = %ChartLegend_Bottom;
&LegendArray = CreateArray("Architect", "Engineer", "Director", "Manager",
   "Senior Director", "Vice-President");
&LegendImgArray = CreateArray("PT_ACECUBE", "PT_WF_PERSON","PT_STATUS_ALERT_ICN",
   "PT_STATUS_ASSIGNED_ICN", "QE_SUCCSORGCHART", "QE_KPERSON");
&rbRatingBoxChart.SetLegend(&LegendArray);
&rbRatingBoxChart.SetLegendImg(&LegendImgArray);
```

7. Set the node data rowset and record.

   The chart refreshes when the SetRBNodeData method is called.

```
/* Set the node data rowset and the node record. */
&rbRatingBoxChart.SetRBNodeData(&rsGrid);
&rbRatingBoxChart.SetRBNodeRecord(Record.Record. QE_RATEBOX_DR);
```

8. Add FieldChange PeopleCode on the fields PTXASISRATINGS and PTYAXISRATINGS if you want to perform other logic after the drag and drop operation.

   This example shows a call to the SetRBNodeData function to redraw the chart the user changes the values in either of these two fields in the component:

```
/* FieldChange PeopleCode for fields PTXAXISRATINGS and PTYAXISRATINGS*/
&rbRatingBoxChart.SetRBNodeData(&rsGrid);
```

Creating a Chart Using an iScript

You can also create a Chart class chart or a Gantt class chart using an iScript. The following example creates a chart, populates it, and then sends the chart URL to a response object. The complete code sample is shown.

Note. You cannot create a chart from OrgChart or RatingBoxChart using an iScript.

```
Function IScript_GetChartURL()

   Local Chart &cChart;
   Local Rowset &rsRowset;
   Local string &sMap;
   Local string &sURL;

   &cChart = CreateObject("Chart");

   &rsRowset = CreateRowset(Record.QE_CHART_RECORD);
   &rsRowset.Fill("where QE_CHART_REGION= :1", "MIDWEST");
   &cChart.SetData(&rsRowset);

   &cChart.Width = 400;
   &cChart.Height = 300;

   &cChart.SetDataYAxis(QE_CHART_RECORD.QE_CHART_SALES);
   &cChart.SetDataXAxis(QE_CHART_RECORD.QE_CHART_PRODUCT);
   &cChart.SetDataSeries(QE_CHART_RECORD.QE_CHART_REGION);

   &cChart.HasLegend = True;
   &cChart.LegendPosition = %ChartLegend_Right;

   &sURL = %Response.GetChartURL(&cChart);
   &sMap = &cChart.ImageMap;

   %Response.Write("<HTML><IMG SRC=");
   %Response.Write(&sURL);
   %Response.Write("  USEMAP=#THEMAP></IMG><MAP NAME=THEMAP>");
   %Response.Write(&sMap);
   %Response.Write("</MAP></HTML>");

End-Function;
```

See *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," CreateObject.

Complete these steps to create a chart using an iScript:

1. Create the chart object.

   For this example, no chart control is available on a page to be referenced. To create the chart object, use the CreateObject function. The string passed in to the function must be the name of the class you are instantiating. For instance, to instantiate a chart from the Chart class, pass the string "Chart".

   ```
   &cChart = CreateObject("Chart");
   ```


2. Create a rowset for the chart data and set the chart data.

   The CreateRowset function creates a standalone rowset data structure. Use the Fill method to populate the empty rowset with data. The SetData method associates the rowset data with the chart.

   ```
   &rsRowset = CreateRowset(Record.QE_CHART_RECORD);
   &rsRowset.Fill("where QE_CHART_REGION= :1", "MIDWEST");
   &cChart.SetData(&rsRowset);
   ```

3. Set the height and width of the chart.

   Because the chart is not associated with a chart control on a page, you have to specify the size of the chart image to be generated using the Height and Width properties. The unit of measurement for both of these properties is pixels.

   ```
   &cChart.Width = 400;
   &cChart.Height = 300;
   ```

4. Set the data axes, series, and legend for the chart.

   As with all charts, you must set the data axes. If necessary for your data, also set the data series and the legend.

   ```
   &cChart.SetDataYAxis(QE_CHART_RECORD.QE_CHART_SALES);
   &cChart.SetDataXAxis(QE_CHART_RECORD.QE_CHART_PRODUCT);
   &cChart.SetDataSeries(QE_CHART_RECORD.QE_CHART_REGION);

   &cChart.HasLegend = True;
   &cChart.LegendPosition = %ChartLegend_Right;
   ```

5. Generate the URL for the chart.

   Use the GetChartURL method of the Response class to generate the URL that will reference the chart data.

   ```
   &sURL = %Response.GetChartURL(&cChart);
   ```

6. Generate the image map.

   Use the ImageMap property to generate the image map for the chart. The Response object uses this value to draw the map.

   ```
   &sMap = &cChart.ImageMap;
   ```

7. Create the response chart.

   Use the Write Response class method to generate the chart. Note that first, the data is set using the URL generated with GetChartURL, and then the image map is used.

   ```
   %Response.Write("<HTML><IMG SRC=");
   %Response.Write(&sURL);
   %Response.Write("  USEMAP=#THEMAP></IMG><MAP NAME=THEMAP>");
   %Response.Write(&sMap);
   %Response.Write("</MAP></HTML>");
   ```

Note. If one or more labels does not render in the chart, increase the width of the chart, use shorter label text, or reduce the data set being charted.

Chapter 12

Component Interface Classes

This chapter provides an overview of Component Interface class and discusses the following topics:

- Life cycle of a Component Interface.
- Declaring a Component Interface object.
- Scope of a Component Interface object.
- Implementing a Component Interface object.
- Component Interface methods and timeouts.
- Traversing a Component Interface and using data collections.
- Working with effective-dated data.
- Reusing existing code.
- Component Interface Reference.

Understanding Component Interface Class

Component Interfaces are the focal points for externalizing access to existing PeopleSoft components. They provide realtime synchronous access to the PeopleSoft business rules and data associated with a component outside the PeopleSoft online system. Component Interfaces can be viewed as "black boxes" that encapsulate PeopleSoft data and business processes, and hide the details of the structure and implementation of the underlying page and data.

Component Interfaces are one of the many APIs that PeopleSoft provides for enabling integration with other systems.

A Component Interface maps to one, and only one, PeopleSoft component. The *Component Interface object*, instantiated from a session object, is created at runtime to access the data specified by the Component Interface.

When you instantiate a Component Interface object:

- All the PeopleCode programs associated with the record fields, pages, component, and so on, and
- The runtime component processor still perform all of the work that they do in the online environment.

The exceptions are any GUI manipulation found in a PeopleCode program, and search dialog specific processing.

Component Interfaces are programmable through a C interface, an OLE/COM or C/C++ interface, and through PeopleCode. Application Engine programs, message notification programs, or any other PeopleCode programs are able to use Component Interfaces.

Like a component, you create the structure of a Component Interface in Application Designer, then at runtime, you populate the structure with data. This document is concerned with the runtime portion of a Component Interface.

When you populate a Component Interface with data, the first thing you fill out are its keys, as you would in a component. These can be keys for getting an existing instance of the data or for creating a new instance of the data.

In addition to keys, a Component Interface is composed of *properties* and *methods.*

- Component Interface *properties* provide access to the data in a component buffer.
- Component Interface *methods* are functions that can be called to perform operations on a Component Interface.

There are two types of both methods and properties: standard and user-defined. Standard properties and methods are provided automatically when you create a Component Interface. They perform operations common to all Component Interfaces, such as indicating what mode to operate the Component Interface, saving, or creating a Component Interface. User-defined properties are the specific record fields that an application developer has chosen to expose to an external system with the Component Interface. User-defined methods are PeopleCode programs that an application developer can write to perform operations on a Component Interface. Each is specific to that Component Interface.

You can instantiate a Component Interface object only from a session object. Through the session object you can control access to the Component Interface, check for errors, control the runtime environment, and so on.

See Also

Chapter 12, "Component Interface Classes," Reusing Existing Code, page 611

Chapter 38, "Session Class," page 2179

PeopleTools 8.51 PeopleBook: PeopleSoft Component Interfaces

Life Cycle of a Component Interface

At runtime, there are certain things you want to do with a Component Interface, such as getting an instance of it, populating it with data, and so on. The following is an overview of this process. These steps are expanded in other sections.

1. Execute the GetCompIntfc method on the PeopleSoft session object to get a Component Interface.
2. Set the key values for the Component Interface object. If the keys you specify don't uniquely describe a component (partial keys), proceed to step 3. If the keys uniquely describe a component, skip to step 4.

3. Do *one* of the following:

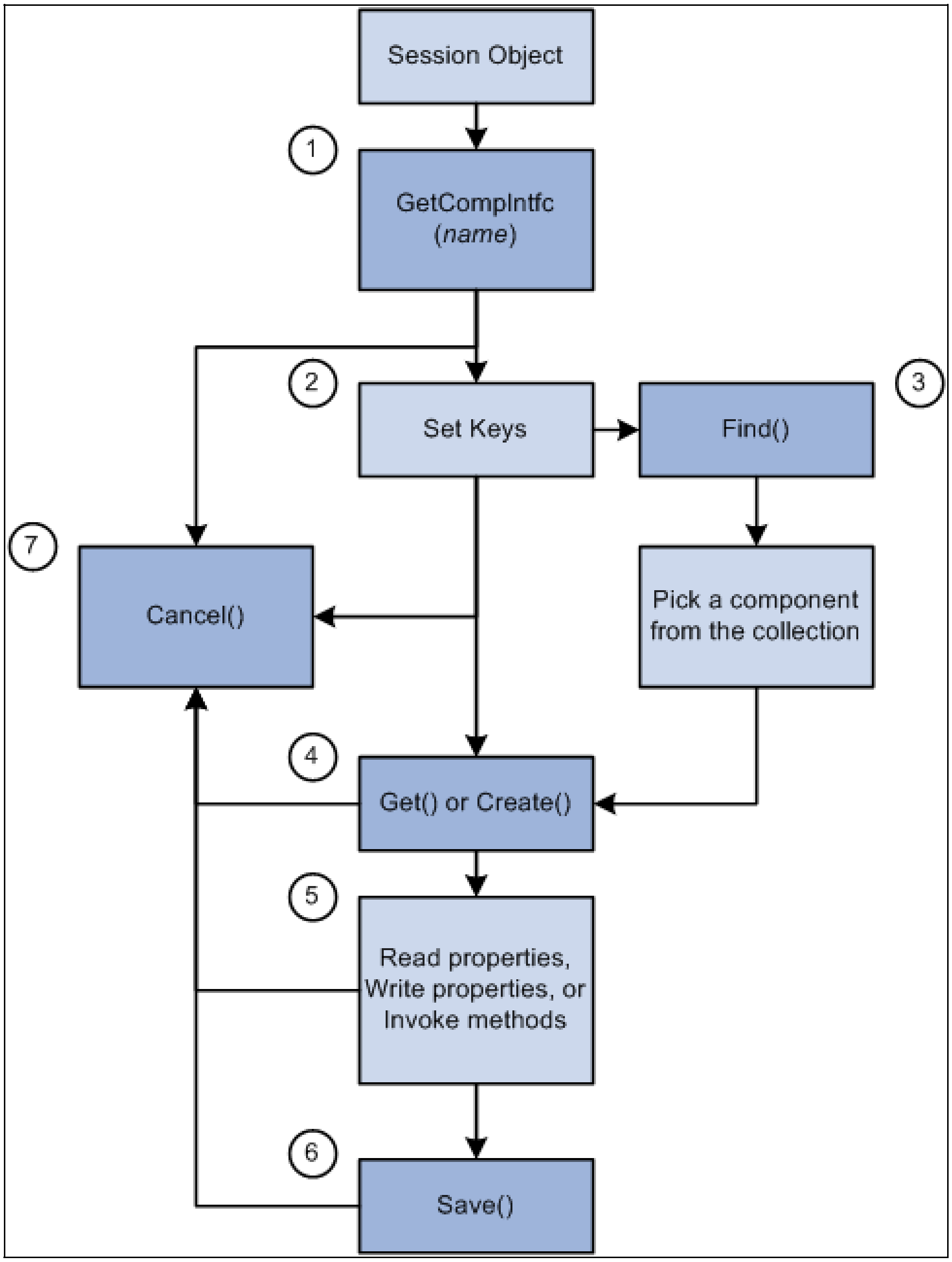
   - Execute the Find standard method on the Component Interface. This returns a collection of Component Interfaces with their key values filled out. The user can then select the unique Component Interface they want.

   - Execute either the Get or Create standard method to instantiate the Component Interface (populate it with data.)

4. Get property values, set property values, or execute user-defined methods. Setting a property value will fire the standard PeopleSoft business rules for the field associated with the property (any PeopleCode program associated with FieldChange, RowInsert, and so on.)

5. Execute the Save standard method to fire the standard PeopleSoft save business rules (any PeopleCode programs associated with SavePreChange, WorkFlow, and so on.) and commit any changes to the database.

6. At any point, the standard method Cancel can be executed to reset the Component Interface to its state in step 1.



Component interface life cycle

Setting Component Interface Keys

The keys for a Component Interface are based on the key fields of the underlying component. There can be different types of keys for a Component Interface.

- CREATEKEYS: A list of the primary key fields of the search record specified to be used in Add mode for the component. This list is automatically generated.
- GETKEYS: A list of the primary (required) key fields on the search record. This list is automatically generated.

- FINDKEYS: A list of primary and alternate key fields on the primary search record for the component.

If the Component Interface has CREATEKEYS, these are the keys you must set before you execute the Create() method to create a new instance of the data. If the Component Interface doesn't have CREATEKEYS, use the GETKEYS to specify a new instance of the data.

Use either the GETKEYS or FINDKEYS to specify an existing instance of the data.

To set key values, use the field names listed under GETKEYS, CREATEKEYS or FINDKEYS like properties on the Component Interface object. The following example sets the CREATEKEYS values to create a new instance of the data.

```
&MYCI = GetCompIntfc(CompIntfc.EXPRESS_ISSUE);
&MYCI.BUSINESS_UNIT = "H01B";
&MYCI.INTERNAL_FLG = "Y";
&MYCI.ORDER_NO = "NEXT&rsquo;;
&MYCI.Create();
```

Standard and User-Defined Component Interface Methods

Every Component Interface comes with a standard set of methods:

- Cancel()
- Create()
- Find()
- Get()
- Save()

Use these methods during runtime to affect the data of the Component Interface.

The application developer can, at design time, disable any of these methods for the Component Interface.

In addition, an application developer can write their own methods. These methods are written as Functions using Component Interface PeopleCode. For example, suppose you wanted to be able to copy an instance of Component Interface data. You might write your own Clone method.

Note. User-defined method names must *not* be named *GetPropertyName.* The C header for Component Interfaces creates functions with that name so you can access each property. If you create your own *GetPropertyName* functions, you receive errors at runtime. User-defined methods can take only simple types of arguments (such as number, character, and so on) because user-defined methods can be called from C/C++ or COM as well as from PeopleCode. PeopleCode can use more complex types (like rowset, array, record, and so on), but these types of arguments are unknown to C/C++ or COM.

See Also

Chapter 12, "Component Interface Classes," Component Interface Class Methods, page 615

PeopleTools 8.51 PeopleBook: PeopleSoft Component Interfaces

PeopleTools 8.51 PeopleBook: PeopleSoft Component Interfaces, "Developing Component Interfaces," Creating User-Defined Methods

Accessing Component Interface Standard Properties

Every Component Interface comes with a standard set of properties. These properties can be divided as follows:

Properties that affect how the Component Interface is executed

The following properties affect how a component interface is executed:

- EditHistoryItems
- GetHistoryItems
- InteractiveMode

These properties must be set before the Component Interface is populated with data. That is, you must set these properties before you use the Get or Create methods.

EditHistoryItems and GetHistoryItems work together to determine how data is accessed:

- If EditHistoryItems is False (the default) and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in update/display All mode. This means all history rows are returned, however, you can edit rows only with a date set in the future.
- If EditHistoryItems is True and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in correction mode. This means all history rows are returned, and you can edit them.
- If GetHistoryItems is False, you access the data in the Component Interface in a similar manner as if you were accessing a component in update mode. The EditHistoryItems has no effect when GetHistoryItems is False.

  InteractiveMode causes the Component Interface to emulate an online component. For example, if you set a value for a field in a Component Interface and you have set InteractiveMode to True, then any FieldChange PeopleCode programs associated with that field fire as soon as you set that value.

Properties that return information about the structure of the Component Interface

The following properties return information about the structure of the Component Interface:

- CreateKeyInfoCollection
- FindKeyInfoCollection

- GetKeyInfoCollection
- PropertyInfoCollection

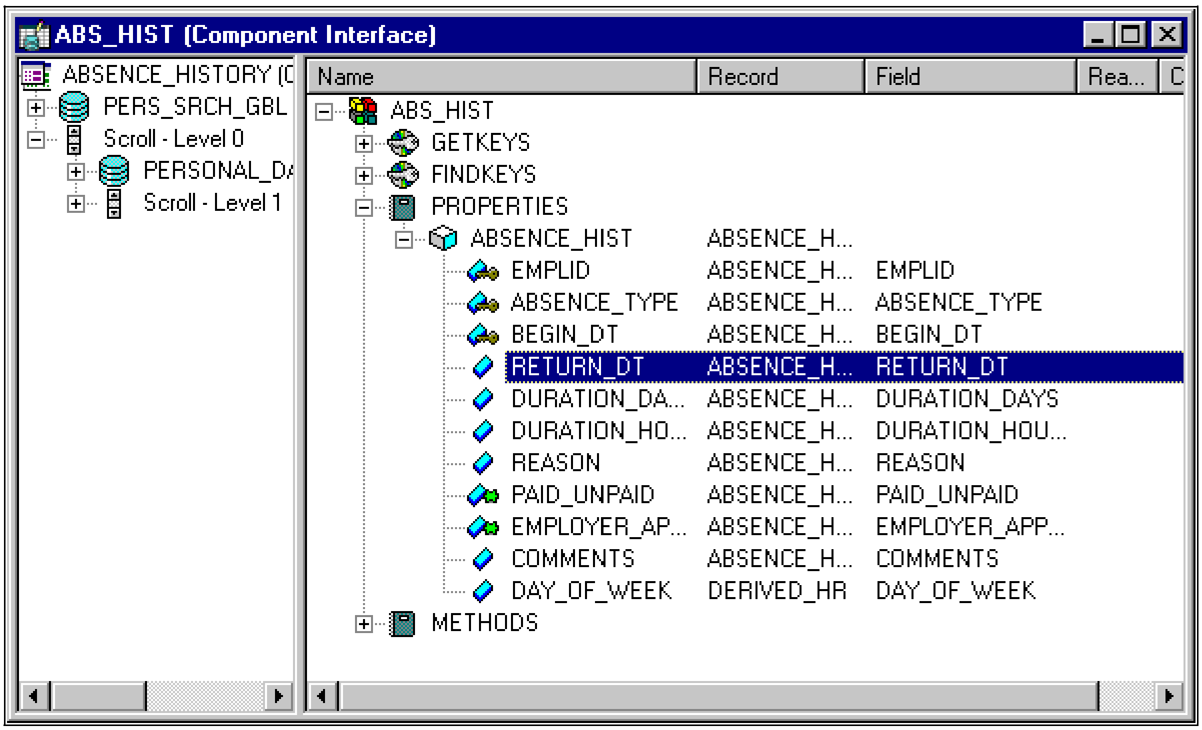
See Also

Chapter 12, "Component Interface Classes," Component Interface Class Properties, page 632

Accessing User-Defined Component Interface Properties

Every user-defined property in a Component Interface *definition* can be used like a property on the *object* instantiated from that Component Interface at runtime.

For example, the following Component Interface definition has RETURN_DT defined as one of its properties.



RETURN_DT Component Interface property highlighted

At runtime, you can use PeopleCode to assign a value to that property (field), or to access the value of that field.

```
&MYCI.RETURN_DT = "05/05/2000";

/*  OR  */

&DATE = &MYCI.RETURN_DT;
```


Data Type of a Component Interface Object

Component Interfaces are declared as type ApiObject. For example,

```
Local ApiObject &MYCI;
```

Note. Component Interface objects can be declared only as Local.

Scope of a Component Interface Object

A Component Interface can be instantiated from PeopleCode, from a Visual Basic program, from Java, COM and C/C++.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

If you're instantiating a Component Interface from an online page, after you finish working with the component, you may want to refresh your page. The Refresh method, on a rowset object, reloads the rowset (scroll) using the current page keys. This causes the page to be redrawn. GetLevel0().Refresh() refreshes the entire page. If you want a particular scroll only to be redrawn, you can refresh just that part.

Considerations Using Component Interfaces and Related Languages

If you update a field using a Component Interface through an external program (such as Java, VB, and so on) the Related Language Table gets updated, even if they are using a base language of ENG.

Considerations Using Component Variables With Component Interfaces

A component variable is scoped locally to its component or its Component Interface. This means that you cannot use a component variable to share data between a component and a Component Interface. In order to share information between components, use global variables.

If your page (component) calls a Component Interface (using the existing session), that in turn initializes a component variable, that component variable is shared with the calling component. When the Component Interface is closed, the component variable is no longer in scope.

When the calling component is closed, any component scoped variables created by it or by the Component Interface go out of scope.

See Also

Chapter 36, "Rowset Class," Refresh, page 2153

PeopleTools 8.51 PeopleBook: PeopleSoft Component Interfaces

Implementing a Component Interface

After you create your Component Interface definition, you can use PeopleCode to access it. This PeopleCode can be long and complex. Rather than write it directly, you can drag and drop the Component Interface definition from Application Designer Project View into an open PeopleCode edit pane. Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify.

You can also access your Component Interface using COM. You can automatically generate a Visual Basic or a C template, similar to the PeopleCode template, to get you started.

The following are the usual actions that you perform with a Component Interface:

- Create a new instance of data
- Get an existing instance of data
- Retrieve a list of instances of data

The following procedures cover each of these actions in more detail.

Another standard action is inserting or deleting a row of data (an item). This involves traversing a Component Interface (going from level zero to level one, from level one to level two, and so on) and accessing data collections.

You may want to work with effective-dated information. There are several properties you can set to allow you to do this.

In addition to these standard actions, you can also look at the structure of a Component Interface.

See Also

Chapter 12, "Component Interface Classes," Create a New Instance of Data Example, page 654

Chapter 12, "Component Interface Classes," Getting an Existing Instance of Data Example, page 657

Chapter 12, "Component Interface Classes," Retrieving a List of Instance of Data Example, page 659

Chapter 12, "Component Interface Classes," Traversing a Component Interface and Using Data Collections, page 608

Chapter 12, "Component Interface Classes," Working With Effective-Dated Data, page 609

Chapter 12, "Component Interface Classes," Accessing the Structure of a Component Interface, page 644

Component Interface Methods and Timeouts

The file pstools.properties controls when a Component Interface method times out. If you're having problems with methods timing out, you may want to change the values in this file. This file is located in the default directory for the application you're running. If this file isn't in this directory, copy it into the directory before you make your changes to it.

Traversing a Component Interface and Using Data Collections

The data in a Component Interface can be contained in a hierarchy: like the page it's built on, there may be data at level zero, level one, level two, and so on.

Each level of data in a Component Interface is known as a collection, such as:

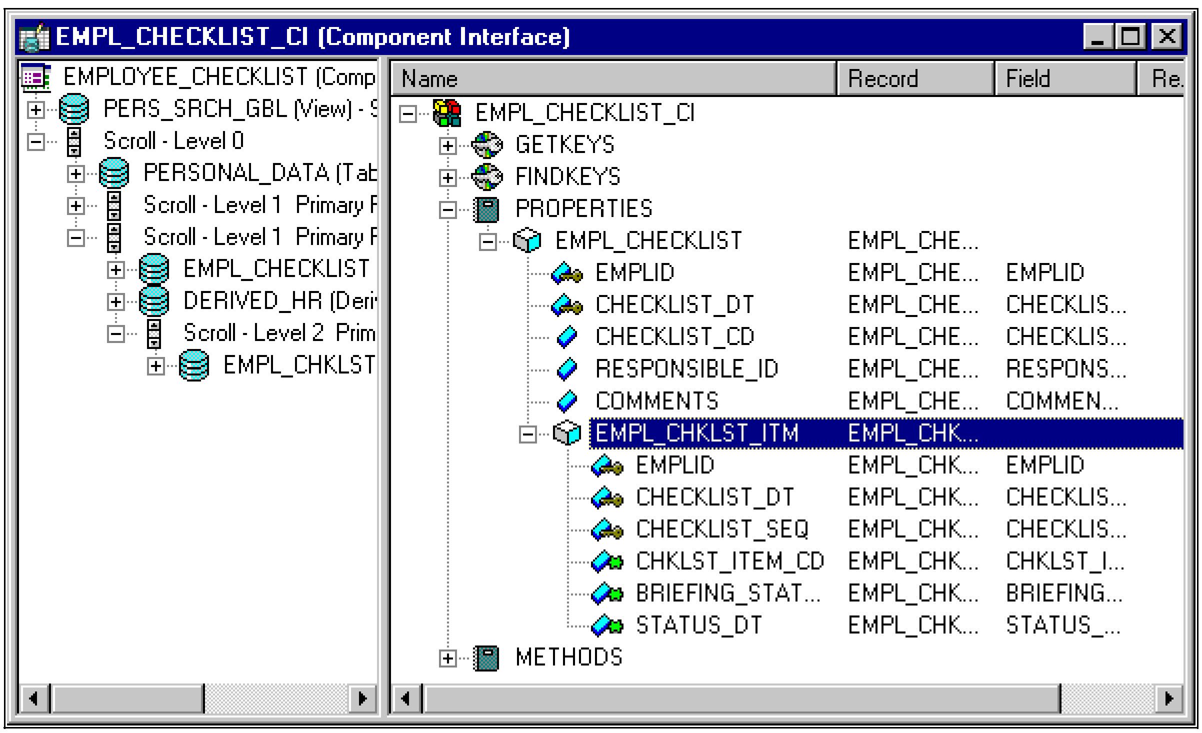
```
level zero

  -- level one (Collection)

    -- Level 2 (Collection)
```

A *collection* is a set of similar things, like a group of already existing Component Interfaces. Most collections have the same standard properties and methods, with some additional ones specific to that collection. For example, all collections have the property *Count*, which indicates how many items are in that collection, but only a data collection has the method *ItemByKeys.*

A *data collection* is the collection of data, available at runtime or during test mode, associated with a particular scroll (or record.) The data collection object returns information about every *row of data* (item) that is returned for that record at runtime.

To access the level two collection, in general, you could use the following:

```
&Level_1 = &CI.Level_1;
&Level_1_Item = &Level_1.Item(ItemNumber);

&Level_2 = &Level_1_Item.Level_2;
```



Sample Component Interface with collection highlighted

This example shows a Component Interface with a two-level hierarchy, that is, *two* data collections: EMPL_CHECKLIST and EMPL_CHKLST_ITM. To get a data collection for the first level, use the following code:

```
&Level1 = &MYCI.EMPL_CHECKLIST;
```

To access the secondary scroll (EMPL_CHKLST_ITM) you have to get the appropriate row (item) of the upper level scroll first:

```
&Level1 = &MYCI.EMPL_CHECKLIST;
&Item = &Level1.Item(1);
&Level2 = &Item.EMPL_CHKLST_ITM;
```

Note. Scrolls represent a hierarchical order. You must get the first level row that contains the secondary scroll *before* you can access the secondary scroll.

Every data collection has a set of methods and standard properties.

See Also

Chapter 12, "Component Interface Classes," Inserting or Deleting a Row of Data Example, page 666

Chapter 12, "Component Interface Classes," Data Collection Methods, page 636

Chapter 12, "Component Interface Classes," Data Collection Properties, page 642

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Working With Effective-Dated Data

When you work with effective-dated data, use some combination of the following methods:

- CurrentItem
- GetEffectiveItem

Both these methods return an item, however, there are some differences:

- The CurrentItem method takes no parameters, so it bases the item it returns on the *current system date.*
- The GetEffectiveItem method bases the item it returns on a *user-provided date.* They all perform the same logic, they just use a different date.

Note. To show all history for an effective-dated collection, you must set GetHistoryItems to True before you populate the Component Interface.

See Also

Chapter 12, "Component Interface Classes," CurrentItem, page 636

Chapter 12, "Component Interface Classes," GetEffectiveItem, page 638

Chapter 12, "Component Interface Classes," GetHistoryItems, page 634

Chapter 12, "Component Interface Classes," Inserting Effective-Dated Data Example, page 661

Chapter 12, "Component Interface Classes," Inserting Effective-Dated Data Example Using Visual Basic, page 664

How a Developer Uses GetEffectiveItem

If a user is making an update to an effective-dated record, they don't always want to insert a row at the end.

Suppose the database contained the following data:

EMPLID	*EFFDT*	*SEQNO*
8000	3/1/99	0
8000	7/1/99	0
8000	9/1/99	0
8000	12/1/99	0

Now suppose the user wants to enter info with EFFDT of 11/1/99. If they were looking at a PeopleSoft component, they would visually scan to see where that date falls and then press ALT+7 and ENTER at the row that they want to insert after.

GetEffectiveItem enables you to pass in the correct effective date, instead of having to loop through every item in the collection doing a comparison, looking for the correct item.

Why Can't it Just Go at the End?

The InsertItem method simulates pressing ALT+7 and ENTER online to insert a row in a scroll. Part of the logic that occurs in the Component Processor is that if the scroll is effective-dated, the ALT+7 and ENTER carries the values forward from the previous row. This functionality is still there if you use InsertItem at the end of the collection, but the values may be incorrect.

Reusing Existing Code

One of the advantages of using a Component Interface is that it enables you to reuse existing PeopleCode and business logic. However, a Component Interface isn't exactly equivalent to a component, which means there are a few key areas in which you should expect differences in behavior between a Component Interface and the component it's based on. This section discusses the differences in:

- Search dialog processing
- Event and function behavior

Differences in Search Dialog Processing

When you run a Component Interface, the *SearchInit*,*SearchSave*, and *RowSelect* events don't fire. This means that any PeopleCode associated with these events won't run. The first event to run is *RowInit.*

Differences in PeopleCode Event and Function Behavior

PeopleCode events and functions that relate exclusively to the page interface (the GUI) and online processing can't be used by Component Interfaces. These include:

- Menu PeopleCode and pop-up menus.

  The ItemSelected and PrePopup PeopleCode events are not supported. In addition, the DisableMenuItem, EnableMenuItem, and HideMenuItem functions aren't supported.

- Transfers between components, including modal transfers.

  The DoModal, EndModal, IsModal, Transfer, TransferPage, DoModalComponent, TransferNode, TransferPortal, and IsModalComponent functions cannot be used.

- Cursor position.

  SetControlValue cannot be used.

- WinMessage cannot be used.
- Save in the middle of a transaction.

  DoSave cannot be used.

- The page Activate event cannot be used.

When executed using a component interface, these functions do nothing and return a default value. In addition, using the Transfer function terminates the current PeopleCode program.

For the unsupported functions, you should put a condition around them, testing whether there's an existing Component Interface.


```
If %ComponentName Then
   /* process is being called from a Component Interface */
   /* do CI specific processing */
Else
   /* do regular processing */
   .  .  .
End-if;
```

Using %Menu Conditions

If you associate a Component Interface definition with a menu in Application Designer, a PeopleCode program that conditionally checks for that menu runs when the Component Interface is executed. For example, suppose you associate a Component Interface with the MP_HR_MENU. The following PeopleCode program, if called from that Component Interface, executes:

```
If %Menu = MP_HR_MENU Then
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Component Interfaces, "Developing Component Interfaces," Associating Component Interfaces with Menus

Session Class Methods

Component Interfaces don't have any built-in functions. They are instantiated from the Session class.

This section discusses the Session class methods used to instantiate Component Interfaces. The methods are discussed in alphabetical order.

FindCompIntfcs

Syntax

FindCompIntfcs([*ComponentName*])

Description

The FindCompIntfcs method, used with the session object, returns a reference to a Component Interface collection, filled with zero or more Component Interfaces. The *ComponentName* parameter takes a string value. You can use a partial value to limit the set of Component Interfaces returned. You can also specify a null value, that is, two quotation marks with no space between them, ("") to return the entire list of Component Interfaces available.

Using the FindCompIntfcs method is equivalent to using File, Open, and selecting Component Interface in Application Designer.

Example

In the following example, a partial key was used to open all components starting with "APP".

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCICOLL = &MYSESSION.FindCompIntfcs("APP");
&MYCI = &MYCICOLL.First();
```

GetCompIntfc

Syntax

GetCompIntfc([**COMPINTFC.**]*ComponentName*)

Description

The GetCompIntfc method, used with the session object, returns a reference to a Component Interface. *ComponentName* when used by itself takes a string value. If you use COMPINTFC.*componentname* it isn't a string value: it's a constant that automatically is renamed in your code if the Component Interface definition is ever renamed. You must specify an existing Component Interface, otherwise you receive a runtime error.

Example

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.PERSONAL_DATA_BC);
```

The following example uses the '@' operator to dynamically call a Component Interface from PeopleCode.

```
&MYCI = &MYSESSION.GetCompIntfc(@("CompIntfc." | &CIname));
```

Component Interface Collection Methods

A Component Interface collection is a list of the available Component Interfaces. An equivalent list is generated by starting Application Designer, selecting File, Open, Component Interface.

You get a Component Interface collection by using the FindCompIntfcs method with a session object.

First

Syntax

First()

Description

The First method returns the first Component Interface in the Component Interface collection object executing the method. This returns the structure of the Component Interface with only the key fields filled in. The rest of the data is not present. To populate the Component Interface with data, you must set the key values and use the Get method.

Example

```
&MYCI = &CICOLLECTION.First();
```

Item

Syntax

Item*(number)*

Description

The Item returns the Component Interface that exists at the number position in the Component Interface collection executing the method. This returns the structure of the Component Interface with only the key fields filled in. The rest of the data is not present. To populate the Component Interface with data, you must set the key values and use the Get method. *number* takes a number value.

Example

```
For &I = 1 to &COLLECTION.Count;
&MYCI = &COLLECTION.Item(&I);
/* do processing */
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next Component Interface in the Component Interface collection object executing the method. You can use this method only after you have used either the First or Item methods: otherwise the system doesn't know where to start. This returns the structure of the Component Interface with only the key fields filled in. To populate the Component Interface with data, you must set the key values and use the Get method.

Example

```
&MYCI = &COLLECTION.Next();
```

Component Interface Collection Property

This section discusses the Count property.

Count

Description

This property returns the number of Component Interfaces in the Component Interface collection, as a number.

This property is read-only.

Example

```
&COUNT = &MYCI_COLLECTION.Count;
```

Component Interface Class Methods

This section discusses the Component Interface class methods. The methods are discussed in alphabetical order.

Cancel

Syntax

Cancel()

Description

The Cancel method cancels the instance of the Component Interface object executing the method, rolling back any changes that were made. This sets the Component Interface state to the same state it was in immediately after it was created by GetCompIntfc. This closes the Component Interface.

Parameters

None.

Returns

A Boolean value: True if component was successfully cancelled, False otherwise.

CopyRowset

Syntax

`CopyRowset``(`*`&Rowset`*` [, `*`InitialRow`*`] [, `*`record_list`*`]);`

Where *record_list* is a list of record names in the form:

```
[RECORD.source_recname1, RECORD.target_recname1
[, RECORD.source_recname2, RECORD.target_recname2]].  .  .
```

Description

The CopyRowset method copies the specified rowset object to the Component Interface executing the method, copying like-named record fields and data collections (child rowsets) at corresponding levels. If pairs of source and destination record names are given, these are used to pair up the records and data collections *before* checking for like-named record fields.

CopyRowset uses the Page Field order when copying properties. This helps ensure that dependent fields are set in the required order.

Note. This method works in PeopleCode only. This method uses the names of the records in the collection, *not* the name you may give the collection when you create the Component Interface.

If there are blanks in *source* rowset or record, they are copied over to the Component Interface only if the field's IsChanged property is set to True. Otherwise blanks are *not* copied.

Use this method when you are using a Component Interface to verify the data in your application message.

Note. The structure of the rowset you're copying data from must exactly match the existing rowset structure, with the same records at level zero, 1, 2, and so on. CopyRowset is intended to be used with a notification process, that is, with message data. As all notifications work only in two-tier mode, CopyRowset works only in two-tier mode.

Parameters

Parameter	Description
&Rowset	Specify an existing, instantiated rowset object that contains data.
InitialRow	Specify the initial transaction row to begin with. This parameter provides a quick way to loop through a rowset that has multiple level zero rows. The default value for this parameter is 1.
record_list	Specify a list of source and target record names. All *source_recnames* are records being copied *from*, in the rowset object being copied from. All *target_recnames* are records being copied *to*, in the Component Interface being copied to.

Returns

None.

Example

The following example would be in your notification PeopleCode. The Exit(1) causes all changes to be rolled back, and the message is marked with the status ERROR so you can correct it.

```
Local message &MSG;
Local ApiObject &SESSION;
Local ApiObject &PO;
Local rowset &ROWSET;

&MSG = %IntBroker.GetMessage();
&ROWSET = &MSG.GetRowset();

&SESSION = %Session;

If &SESSION <> NULL Then
/* Session connected correctly */
/* Set key values to create component */
&PO = &SESSION.GetCompIntfc(COMPINTFC.PO);
&PO.BU = &MSG(&I).PO_HDR.BU.Value;
&PO.PO_ID = &MSG(&I).PO_HDR.PO_ID.Value;
&PO.Create();

&PO.CopyRowset(&ROWSET);
If NOT (&PO.Save()) Then
   Exit(1);
End-if;
Else
/* do error processing */
End-If;
```

The following example loops through all the transactions of a message rowset.

```
Local message &MSG;
Local ApiObject &SESSION;
Local ApiObject &CI;
Local rowset &ROWSET;

&MSG = %IntBroker.GetMessage();
&ROWSET = &MSG.GetRowset();


&SESSION = %Session;
If &SESSION <> Null Then)
   &CI = &SESSION.GetCompIntfc(CompIntfc.VOL_ORG);
   &I = 0;
   While (&I < &ROWSET.RowCount)
      &I = &I + 1;
      &CI.VOLUNTEER_ORG = &ROWSET.GetRow(&I).VOLNTER_ORG_2.  VOLUNTEER_ORG.Value;
      If &CI.Create() Then
         If &CI.CopyRowset(&ROWSET, &I, Record.VOLNTER_ORG_TBL, Record.⇒
VOLNTER_ORG_TBL) Then

         /* App specific processing */

            If Not &CI.Save() Then
               Winmessage("Save Failed");
               /* Other app specific processing */
            End-If;
         End-If;
      End-If;
   &CI.Cancel();
   End-While;
End-If;
```

CopyRowsetDelta

Syntax

CopyRowsetDelta(*&Rowset* [, *InitialRow*] [, *record_list*]);

Where *record_list* is a list of record names in the form:

```
[RECORD.source_recname1, RECORD.target_recname1
[, RECORD.source_recname2, RECORD.target_recname2]].  .  .
```

Description

The CopyRowsetDelta method copies the changed rows in the specified rowset object to the Component Interface executing the method, copying like-named record fields and data collections (child rowsets) at corresponding levels. If pairs of source and destination record names are given, these are used to pair up the records and data collections *before* checking for like-named record fields.

Note. This method works in PeopleCode only. This method uses the names of the records in the collection, *not* the name you give the collection when you create the Component Interface.

If there are blanks in *source* rowset or record, they are copied over to the Component Interface only if the field's IsChanged property is set to True. Otherwise blanks are *not*copied.

You will generally use this method with a Component Interface to verify data from a message.

Note. CopyRowsetDelta is intended for a notification process, that is, with message data. As all notifications work only in two-tier mode, CopyRowsetDelta works only in two-tier mode.

If the rowset you're copying from is a message rowset, the CopyRowsetDelta method uses the AUDIT_ACTN field in the PSCAMA table in the message to know whether the row is to be inserted, updated, or deleted inside the Component Interface.

If the rowset you're copying from *is not* a message rowset, that rowset must have the same structure as a message, that is, it must have a PSCAMA record with an AUDIT_ACTN field on every level of the rowset.

Warning! CopyRowsetDelta uses a record's keys to locate the target row to change for all audit actions other than Add. CopyRowsetDelta actions (other than Add) therefore work only on rowsets that have keys *that uniquely identify all rows in the rowset.* Rowsets that do *not*distinguish between rows using a key field will be updated in an unpredictable fashion.

Considerations Using CopyRowsetDelta with Effective-Dated Rowsets

If a message data row inserted using a PSCAMA Audit action of "A" belongs to an effective dated scroll containing child scrolls, the insertion of the parent row causes child rows of the previous effective-dated row to be copied over, and their effective date is updated with that of the inserted parent.

If the message also contains a child row being inserted with a PSCAMA Audit action of "A", the component interface being populated will end up having two child rows: the one inserted as part of the Effective-dated processing and the one inserted using the PSCAMA Audit action "A" in the message.

Parameters

Parameter	***Description***
&Rowset	Specify an existing, instantiated rowset object that contains data.
InitialRow	Specify the initial transaction row to begin with. This parameter provides a quick way to loop through a rowset that has multiple level zero rows. The default value for this parameter is 1.
record_list	Specify a list of source and target record names. All *source_recnames* are records being copied *from*, in the rowset object being copied from. All *target_recnames* are records being copied *to*, in the Component Interface being copied to.

Returns

None.

Example

The following PeopleCode would exist in your notification process. The Exit(1) causes all changes to be rolled back, and the message is marked with the status ERROR so you can correct it.


```
Local Message &MSG;
Local ApiObject &SESSION;
Local ApiObject &PO;
Local Rowset &ROWSET;

&MSG = %IntBroker.GetMessage();
&ROWSET = &MSG.GetRowset();

&SESSION = %Session;

If &SESSION <> Null Then
/* Session connected correctly */
/* Set key values to get component */
&PO = &SESSION.GetCompIntfc(COMPINTFC.PO);
&PO.BU = &MSG(&I).PO_HDR.BU.Value;
&PO.PO_ID = &MSG(&I).PO_HDR.PO_ID.Value;
&PO.Get();


&PO.CopyRowsetDelta(&ROWSET);
If NOT (&PO.Save()) Then
   Exit(1);
End-if;
Else
   /* do error processing */
End-If;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Understanding Supported Message Structures," PSCAMA

CopySetupRowset

Syntax

CopySetupRowset*(&Rowset* [, *InitialRow*] [, *record_list*]);

Where *record_list* is a list of record names in the form:

RECORD.*source_recname,* RECORD.*target_recname*

Description

The CopySetupRowset method is used to copy a setup table application message to a Component Interface. A setup table has the same record at level zero and level one, while a Component Interface has only a single copy of a record. This method copies the contents of the message (at level zero) to the first level collection (level one) of the Component Interface.

The CopySetupRowset method copies *like-named* record fields. If a pair of source and destination record names are given, these are used to pair up the records and data collections *before* checking for like-named record fields.

Note. This method works in PeopleCode only. This method uses the names of the records in the collection, *not* the name you may give the collection when you create the Component Interface.

If there are blanks in *source* rowset or record, they are copied over to the Component Interface only if the field's IsChanged property is set to True. Otherwise blanks are *not* copied.

Note. CopySetupRowset is for a notification process, that is, with message data. As all notifications work only in two-tier mode, CopySetupRowset works only in two-tier mode.

Parameters

Parameter	*Description*
&Rowset	Specify an existing, instantiated rowset object that contains data.
InitialRow	Specify the initial transaction row to begin with. This parameter provides a quick way to loop through a rowset that has multiple level zero rows. The default value for this parameter is 1.
record_list	Specify source and target record names. The *source_recname* is the record being copied *from*, in the rowset object being copied from. The *target_recname* is the record being copied *to*, in the Component Interface being copied to.

Returns

None.

Example

The following example would be in your Notification PeopleCode.


```
Local ApiObject &SESSION;
Local ApiObject &PSMESSAGES;

Local ApiObject &CI;

Local Message &MSG;
Local Rowset &RS;

&SESSION = %Session;
&PSMESSAGES = &SESSION.psmessages;

&MSG = %IntBroker.GetMessage();
&RS = &MSG.GetRowset();

&CI = &SESSION.getcomponent(Component.VOL);

/** Set Business Component Properties **/
&CI.gethistoryitems = True;
/*&CI.InteractiveMode = True;*/
/*&CI.stoponfirsterror = True;*/

For &TRANSACTION = 1 To &RS.RowCount

   &CI.VOLUNTEER_ORG = &RS(&TRANSACTION).VOLNTER_ORG_TBL.VOLUNTEER_ORG.Value;

   /* note: You can achieve better performance if you add code here to check  if⇒
  the keys are the same L0 keys as the last time  through the loop and, if so,skip⇒
  the Get/Create section. */

   If Not &CI.create() Then
      &PSMESSAGES.DeleteAll();
      If Not &CI.get() Then
         /** Check Error Messages **/
         For &I = 1 To &PSMESSAGES.count
            &TYPE = &PSMESSAGES.item(&I).type;
            &TEXT = &PSMESSAGES.item(&I).text;
         End-For;
         Exit(1);

      End-If;
   End-If;

   If Not &CI.CopySetupRowset(&RS, &TRANSACTION) Then
      /** Check Error Messages **/
      For &I = 1 To &PSMESSAGES.count
         &TYPE = &PSMESSAGES.item(&I).type;
         &TEXT = &PSMESSAGES.item(&I).text;
      End-For;
      Exit(1);
   End-If;

   If Not &CI.Save() Then
      /** Check Error Messages **/
      For &I = 1 To &PSMESSAGES.count
         &TYPE = &PSMESSAGES.item(&I).type;
         &TEXT = &PSMESSAGES.item(&I).text;
      End-For;
      Exit(1);
   End-If;

   &CI.Cancel();
End-For;
```


See Also

Chapter 12, "Component Interface Classes," CopySetupRowsetDelta, page 623 and Chapter 12, "Component Interface Classes," CopyRowset, page 616

CopySetupRowsetDelta

Syntax

```
CopySetupRowsetDelta(&Rowset [, InitialRow] [, record_list]);
```

Where *record_list* is a list of record names in the form:

```
RECORD.source_recname, RECORD.target_recname
```

Description

The CopySetupRowsetDelta method is used to copy a setup table with *changed* rows in an application message to a Component Interface. A setup table has the same record at level zero and level one, while a Component Interface has only a single copy of a record. This method copies the contents of the message (at level zero) to the first level collection (level one) of the Component Interface.

Note. This method works in PeopleCode only. CopySetupRowsetDelta copies all the like-named fields from the changed *row* into the message. It is *not* copying only the changed fields.

The CopySetupRowsetDelta method copies *like-named* record fields, in those rows where the IsChanged property is True. If a pair of source and destination record names are given, these are used to pair up the records and data collections *before* checking for like-named record fields.

Note. This method uses the names of the records in the collection, *not*the name you may give the collection when you create the Component Interface.

If there are blanks in *source* rowset or record, they are copied over to the Component Interface only if the field's IsChanged property is set to True. Otherwise blanks are *not*copied.

Note. CopySetupRowsetDelta is for a notification process, that is, with message data. As all notifications work only in two-tier mode, CopySetupRowsetDelta works only in two-tier mode.

Warning! CopySetupRowsetDelta uses a record's keys to locate the target row to change for all audit actions other than Add. CopySetupRowsetDelta actions (other than Add) therefore work only on rowsets that have keys that *uniquely identify all rows in the rowset.* Rowsets that do *not*distinguish between rows using a key field will be updated in an unpredictable fashion.

Parameters

Parameter	***Description***
&Rowset	Specify an existing, instantiated rowset object that contains data.
InitialRow	Specify the initial transaction row to begin with. This parameter provides a quick way to loop through a rowset that has multiple level zero rows. The default value for this parameter is 1.
record_list	Specify source and target record names. The *source_recname* is the record being copied *from,* in the rowset object being copied from. The *target_recname* is the record being copied *to,* in the Component Interface being copied to.

Returns

None.

Example

```
Local ApiObject &SESSION;
Local ApiObject &PSMESSAGES;

Local ApiObject &CI;

Local Message &MSG;
Local Rowset &RS;

&SESSION = %Session;
&PSMESSAGES = &SESSION.psmessages;

&MSG = %IntBroker.GetMessage();
&RS = &MSG.GetRowset();

&CI = &SESSION.getcomponent(Component.VOL);

/** Set Business Component Properties **/
&CI.gethistoryitems = True;
/*&CI.InteractiveMode = True;*/
/*&CI.stoponfirsterror = True;*/

For &TRANSACTION = 1 To &RS.RowCount

   &CI.VOLUNTEER_ORG = &RS(&TRANSACTION).VOLNTER_ORG_TBL.VOLUNTEER_ORG.Value;

   /* note: You can achieve much better performance if you add code here to check ⇒
 if the keys are the same L0 keys as the last time through the loop and, if so,⇒
 skip the Get/Create section. */

   If Not &CI.create() Then
      &PSMESSAGES.DeleteAll();
      If Not &CI.get() Then
         /** Check Error Messages **/
         For &I = 1 To &PSMESSAGES.count
            &TYPE = &PSMESSAGES.item(&I).type;
            &TEXT = &PSMESSAGES.item(&I).text;
         End-For;
         Exit(1);

      End-If;
   End-If;

   If Not &CI.CopySetupRowsetDelta(&RS, &TRANSACTION) Then
      /** Check Error Messages **/
      For &I = 1 To &PSMESSAGES.count
         &TYPE = &PSMESSAGES.item(&I).type;
         &TEXT = &PSMESSAGES.item(&I).text;
      End-For;
      Exit(1);
   End-If;

   If Not &CI.Save() Then
      /** Check Error Messages **/
      For &I = 1 To &PSMESSAGES.count
         &TYPE = &PSMESSAGES.item(&I).type;
         &TEXT = &PSMESSAGES.item(&I).text;
      End-For;
      Exit(1);
   End-If;
```

```
&CI.Cancel();
```

See Also

Chapter 12, "Component Interface Classes," CopySetupRowset, page 620 and Chapter 12, "Component Interface Classes," CopyRowsetDelta, page 618

Create

Syntax

Create()

Description

The Create method associates the Component Interface object executing the method with a new, open Component Interface that matches the key values that were set prior to using the Create method. If there are CREATEKEYS values associated with the Component Interface, these are the values you must set. If there are no CREATEKEYS values, you must set the required GETKEYS values instead. All keys required for creating a new Component Interface must be set before using the Create method, otherwise you receive a runtime error. If you do not use unique key values, (that is, you try to set the keys to values that already exist in the database) you receive a runtime error.

Setting the key values prior to using the Create method is analogous to filling in the key values in the Add dialog for a component when you access it in add mode.

Parameters

None.

Returns

A Boolean value: True if component was successfully created, False otherwise.

Example

```
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.ACTION_REASON);
&MYCI.ACTION = "Additional Job";
&MYCI.ACTION_REASON = "0001";
&MYCI.Create();
```


Find

Syntax

Find()

Description

Note. Find can be used only at level zero within the Component Interface.

You do not have to set values for all the key values. You can use the same wildcards in your Find that you can use in the search dialog, that is, % for one or more characters in a search pattern, and _ (underscore) for exactly one character. In addition, you can use partial values. For example, the following code finds all the data items where the employee ID starts with an "8":

```
&MYCI.Emplid = "8";
&MYDC = &MYCI.Find();
```

This is analogous to using a partial key from the search dialog, opening a component.

After you have a data collection, you can use one of the data collection methods to open the Component Interface.

Parameters

None.

Returns

A collection of Component Interfaces.

See Also

Chapter 12, "Component Interface Classes," Data Collection Methods, page 636

Get

Syntax

Get()

Description

The Get method associates the Component Interface object executing the method with an open Component Interface that matches the key values that were set prior to using the Get method. The key values you must set are the required GETKEYS values for the Component Interface.

Note. To retrieve all the history data for a Component Interface, you must specify the GetHistoryItems property as True *before* you use the Get method. If you want any PeopleCode programs associated with the fields to fire immediately after a value is changed, you must set the InteractiveMode property as True *before* you use the Get method.

After any execution of Get, you should check if there are any errors pending on the session object. In some special circumstances (involving failure of previously cached operations failing after the Get has executed) Get returns True even though the component wasn't retrieved.

Parameters

None.

Returns

A Boolean value: True if component was successfully retrieved, False otherwise.

Example

```
&MYCI.EMPLID = "8001";
&MYCI.Get();
If %Session.ErrorPending Then
   /* Get Unsuccessful, do error processing */
Else
   /* do regular processing */
End-if;
```

GetPropertyByName

Syntax

GetPropertyByName*(string)*

Description

The GetPropertyByName method returns the value of the property specified by *string.* For a collection, it returns a reference to the collection. Generally this function is used only in applications that cannot get the names of the component interface properties until runtime.

Parameters

Parameter	Description
string	The name of the property.

Returns

String. The value of the property.

ApiObject. For a collection. The value of the property.

Example

```
Local ApiObject &oSession, &oCI;
Local array of string &Keys_Arr, &Temp_Arr, &Prop_Arr;
Local string &strCIName, &PropertyValue;
Local number &I, &J, &K;

Function getPropertyValue(&oDataColl As ApiObject, &PropertyName As string)⇒
 Returns string
rem ***** Return property value
   Return &oDataColl.GetPropertyByName(&PropertyName);
End-Function;

Function getCollection1(&collectionName As string)
   Local ApiObject &oL1_DataColl, &oL1_DataItem;
rem ***** Return collection
   &oL1_DataColl = &oCI.GetPropertyByName(&collectionName);
   For &J = 1 To &oL1_DataColl.Count
      &oL1_DataItem = &oL1_DataColl.Item(&J);
      For &K = 1 To &Prop_Arr.Len
         &Temp_Arr = Split(&Prop_Arr [&K], "|");
         If &Temp_Arr [1] = "1" Then
            If &Temp_Arr [3] = "Property" Then
               &PropertyValue = getPropertyValue(&oL1_DataItem,
                   &Temp_Arr [2]);
            Else
               rem ***** Code to Get Collection 2 goes here *****;
            End-If;
         End-If;
      End-For;
   End-For;
End-Function;
```

See Also

Chapter 12, "Component Interface Classes," SetPropertyByName, page 631

Save

Syntax

Save()

Description

Saves any changes that have been made to the data of the Component Interface executing the method. You must save any new Component Interfaces you create before they are added to the database.

The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with SaveEdit, SavePreChange, WorkFlow, and so on.) is executed after you execute this method. If you didn't specify the Component Interface to run in interactive mode, FieldEdit, FieldChange, and so on, also run at this time.

If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, you may want to delete it from the PSMessages collection.

Note. If you're running a Component Interface from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

Parameters

None.

Returns

A Boolean value: True if component was successfully saved, False otherwise.

Example

```
&MYCI.Emplid = "8001";
&MYCI.Get();
&MYCI.CHECKLIST_CD = "00001";
&MYCI.Save;
```

See Also

Chapter 12, "Component Interface Classes," InteractiveMode, page 635

Chapter 38, "Session Class," ErrorPending, page 2188

SetPropertyByName

Syntax

SetPropertyByName(*string, value***)**

Description

The SetPropertyByName method sets the value of the property specified by *string.* Generally this function is used only in applications that cannot set the names of the component interface properties until runtime.

Parameters

Parameter	***Description***
string	The name of the property.
value	The value to which the property is to be set.

Returns

None.

Example

```
Local ApiObject &oSession, &oCI;
Local array of string &Keys_Arr, &Temp_Arr, &Prop_Arr;
Local string &strCIName, &PropertyValue;
Local number &I, &J, &K;

&strCIName = "SDK_BUS_EXP";
&Keys_Arr = CreateArrayRept("", 0);
&Keys_Arr.Push("SDK_EMPLID|" | "8001");

&oSession = %Session;
&oCI = &oSession.GetCompIntfc(@("CompIntfc." | &strCIName));

For &I = 1 To &Keys_Arr.Len
   &Temp_Arr = Split(&Keys_Arr [1], "|");
   &oCI.SetPropertyByName(&Temp_Arr [1], &Temp_Arr [2]);
End-For;
```

See Also

Chapter 12, "Component Interface Classes," GetPropertyByName, page 628

Component Interface Class Properties

In this section, we discuss the Component Interface class properties. The properties are discussed in alphabetical order.

ComponentName

Description

This property returns the name of the Component Interface, as defined in Application Designer, as a string.

This property is read-only.

CreateKeyInfoCollection

Description

This property returns a CompIntfPropInfoCollection collection that contains a CompIntfPropInfoCollection object for every key in CREATEKEYS.

This property is read-only.

Example

```
&CREATEKEYS = &CI.CreateKeyInfoCollection;
```

See Also

Chapter 12, "Component Interface Classes," Accessing the Structure of a Component Interface, page 644

EditHistoryItems

Description

This property works with the GetHistoryItems property to specify what data is accessed, and whether you can edit that data:

- If EditHistoryItems is False (the default) and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in Update/Display All mode. This means all history rows are returned, however, you can edit only rows with a date set in the future.

- If EditHistoryItems is True and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in Correction mode. This means all history rows are returned, and you can edit them.
- If GetHistoryItems is False, you access the data in the Component Interface in a similar manner as if you were accessing a component in Update mode. The EditHistoryItems has no effect when GetHistoryItems is False.

You must set this property to True *before* you execute the Get method.

This property is read-write.

See Also

Chapter 12, "Component Interface Classes," GetHistoryItems, page 634

FindKeyInfoCollection

Description

This property returns a CompIntfPropInfoCollection collection that contains a CompIntfPropInfoCollection object for every key in FINDKEYS.

This property is read-only.

See Also

Chapter 12, "Component Interface Classes," Accessing the Structure of a Component Interface, page 644

GetDummyRows

Description

When a new scroll is inserted on a page, that scroll is displayed even though it has no underlying data. Any scroll that is empty has one dummy row displayed with only the defaults set.

This property is True if the dummy row is to be displayed, False if you do not want to display the dummy row. The default value for this property is True.

This property is read-write.

Example

```
&MyCI.GetDummyRows = False;
```


GetHistoryItems

Description

This property works with the EditHistoryItems property to specify what data is accessed, and whether you can edit that data:

- If EditHistoryItems is False (the default) and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in Update/Display All mode. This means all history rows are returned, however, you can edit only rows with a date set in the future.
- If EditHistoryItems is True and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in Correction mode. This means all history rows are returned, and you can edit them.
- If GetHistoryItems is False, you access the data in the Component Interface in a similar manner as if you were accessing a component in Update mode. The EditHistoryItems has no effect when GetHistoryItems is False.

You must set this property to True *before* you execute the Get method.

This property is read-write.

Example

The following example checks the current status of the mode, then sets the GetHistoryItems and EditHistoryItems properties to True if the mode is Correction mode.

```
If %Mode = "C" Then
   &CI.EditHistoryItems = True;
   &CI.GetHistoryItems = True;
End-if;
```

See Also

Chapter 12, "Component Interface Classes," EditHistoryItems, page 632

GetKeyInfoCollection

Description

This property returns a CompIntfPropInfoCollection collection that contains a CompIntfPropInfoCollection object for every key in GETKEYS.

This property is read-only.

See Also

Chapter 12, "Component Interface Classes," Accessing the Structure of a Component Interface, page 644

InteractiveMode

Description

When this property is set as True, the Component Interface runs the same as a component: that is, any PeopleCode programs associated with FieldChange, RowInsert, and so on, run immediately after you make a change. If this property is set to False, these programs won't run until you execute the Save method.

This property is read-write.

See Also

Chapter 38, "Session Class," ErrorPending, page 2188

PropertyInfoCollection

Description

This property returns a CompIntfPropInfoCollection object for every property that isn't a collection (that is, a scroll.) If the property is a collection (scroll), use the PropertyInfoCollection property to get another collection.

This property is read-only.

See Also

Chapter 12, "Component Interface Classes," Accessing the Structure of a Component Interface, page 644

StopOnFirstError

Description

When this property is set as True, the CopyRowset (or CopyRowsetDelta) method currently executing halts its processing at the first error generated by the Component Interface.

When this property is set as True and InteractiveMode is set as False, processing of queued operations at save time is halted at the first error.

The default value is False.

This property is read-write.

Data Collection Methods

A *data collection* is the collection of data, available at runtime or during test mode, associated with a particular scroll (or record.) The data collection object returns information about every *row of data* (item) that is returned for that record at runtime.

To access a data collection, use the name of the record (scroll) as a property on a Component Interface.

See Also

Chapter 12, "Component Interface Classes," Traversing a Component Interface and Using Data Collections, page 608

CurrentItem

Syntax

CurrentItem()

Description

If the component associated with the Component Interface is effective-dated, CurrentItem returns a reference to the current effective-dated item (row of data). To get a specific item based on a date, use GetEffectiveItem.

If there is no current item, a Null is returned.

Example

```
&MYCD = &MYCI.EMPL_CHKLST_ITM;
&ITEM = &MYDC.CurrentItem();
```

DeleteItem

Syntax

DeleteItem*(number)*

Description

The DeleteItem method deletes the item (row of data) at the position specified by *number*. This parameter takes a number value.

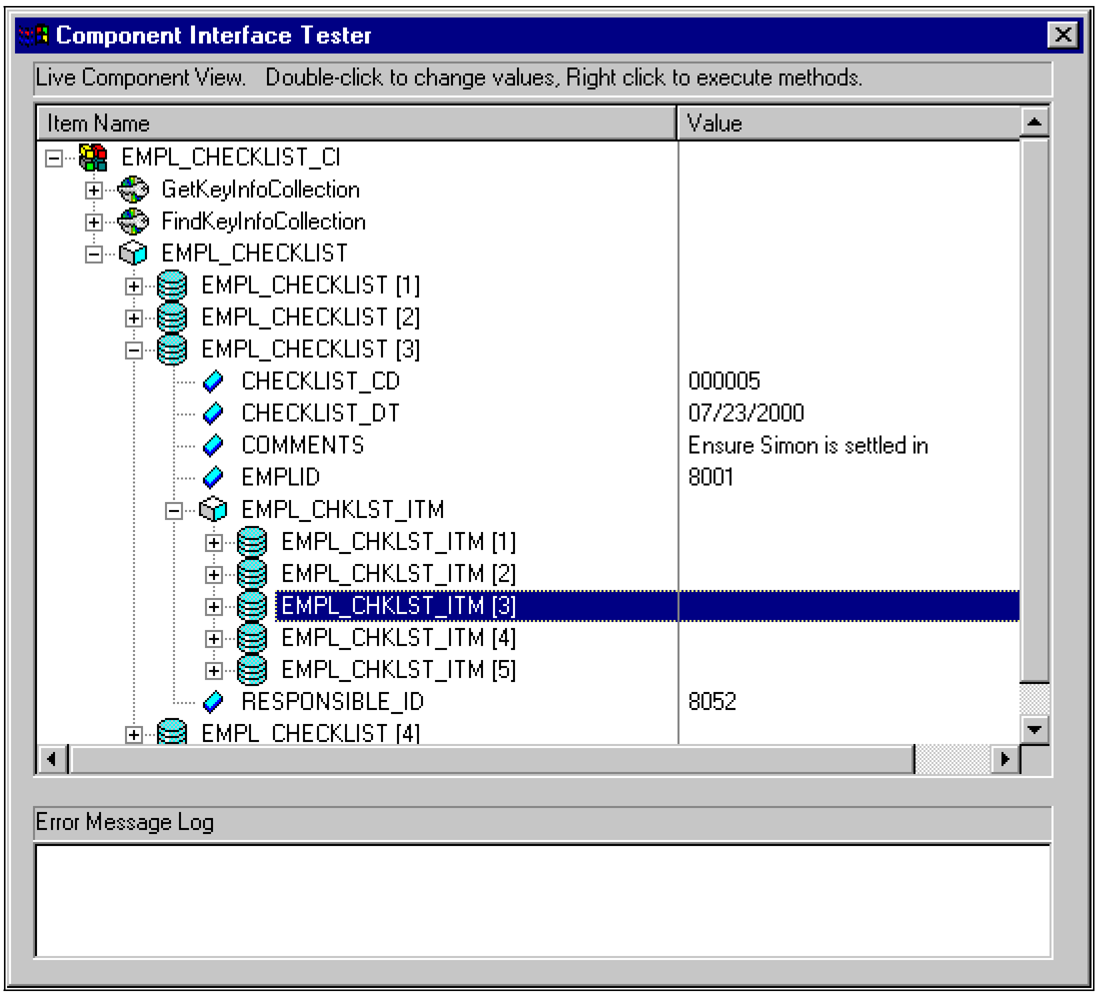
When the DeleteItem method is executed, if there are any RowDelete PeopleCode programs associated with any of the fields, they fire as well, as if the user pressed ALT+8 and ENTER or clicked the DeleteRow icon. However, the programs are executed as if turbo mode was selected. (In turbo mode default processing is performed *only* for the row being deleted.)

If you set the InteractiveMode property to True, any RowDelete PeopleCode runs immediately after you execute DeleteItem. If this property is set to False, any RowDelete PeopleCode runs after you execute the Save method.

The deleted item is not actually deleted from the database until after you use the Save method.

Example

For example, suppose your Component Interface had two scrolls: EMPL_CHECKLIST and EMPL_CHKLST_ITM. The data collection EMPL_CHECKLIST has four items (rows of data.) The data collection EMPL_CHKLST_ITM (under the third item) has five items (rows of data.) If you run this component in the Component Interface Tester, it would look as follows:



EMPL_CHK_BC in Component Interface Tester

To delete the third row in the third item, use the following:


```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHK_BC);
&MYLVL1 = &MYCI.EMPL_CHECKLIST;
&ITEM2 = &MYLVL1.Item(3);
&MYLVL2 = &ITEM2.EMPL_CHKLST_ITM;
&MYLVL2.DeleteItem(3);
&MYCI.Save();
```

GetEffectiveItem

Syntax

GetEffectiveItem*(DateString, SeqNo)*

Description

If the component associated with the Component Interface is effective-dated, GetEffectiveItem returns a reference to the closest effective-dated item (row of data) that is less than or equal to the date specified by the *DateString.* To get an item based on the current effective date, use CurrentItem.

Note. *DateString* takes only a string value. You must convert a date variable into a string before you can use it for *DateString.* You can use the String function to do this.

Parameters

Parameter	*Description*
DateString	Specify the year, month, and day of the effective date that you want to look for. This parameter takes a string value. You can specify the date either as YYYY-MM-DD *or*MM/DD/YY.
SeqNo	Specify the sequence number of the effective date that you want to look for. This parameter takes a number value.

Returns

A reference to an effective-dated item.

Example

```
&MYCD = &MYCI.EMPL_CHKLST_ITM;
&DSTRING = String(&MYDATE);
&ITEM = &MYDC.GetEffectiveItem(&DSTRING, 1);
```


See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," String

GetEffectiveItemNum

Syntax

GetEffectiveItemNum*(DateString, SeqNo)*

Description

If the component associated with the Component Interface is effective-dated, GetEffectiveItemNum returns a reference to the number of the closest effective-dated item (row of data) that is less than or equal to the date specified by the *DateString.* To get an item number based on the current effective date, use CurrentItemNum.

Note. *DateString* takes only a string value. You must convert a date variable into a string before you can use it for *DateString.* You can use the String function to do this.

Considerations for Returning Rows

GetEffectiveItemNum returns a valid row number only when EFFDT is less than or equal to *DateString.* If the value you use for *DateString* pre-dates all the rows in the data collection, this method returns a zero and logs a message in the PSMessages collection.

For example, if 12/01/1990 was the earliest date in the collection, the following would return zero:

```
&NUM = &MYDC.GetEffectiveItemNum("01/01/1900", 1);
```

Parameters

Parameter	*Description*
DateString	Specify the year, month, and day of the effective date that you want to look for. This parameter takes a string value. You can specify the date either as YYYY-MM-DD *or*MM/DD/YYYY.
SeqNo	Specify the sequence number of the effective date that you want to look for. This parameter takes a number value.

Returns

A number. This method returns 0 if no row matching the criteria is found.

Example

```
&MYCD = &MYCI.EMPL_CHKLST_ITM;
&DSTRING = String(&MYDATE);
&ITEMNUM = &MYDC.GetEffectiveItemNum(&DSTRING, 1);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," String

InsertItem

Syntax

InsertItem*(number)*

Description

The InsertItem method inserts the item (row of data) at the position specified by *number*. This parameter takes a number value. You can insert items below only the zero level scroll. If you need to add a new data item, use the Create method instead.

InsertItem adds the new row *after* the current row. If the row has an effective date (EFFDT) or an effective sequence (EFFSEQ), these values are copied into the new row.

If you specify -1 for *number*, InsertItem inserts a new item (row) after the *last* row.

The InsertItem method returns a reference to the newly created item (row of data).

When the InsertItem method is executed, if there are any RowInsert PeopleCode programs associated with any of the fields, they fire also, as if the user pressed ALT+7 and ENTER or clicked the InsertRow icon. However, the programs are executed as if turbo mode was selected. (In turbo mode default processing is performed *only* for the row being inserted.)

If you set the InteractiveMode property to True, any RowInsert PeopleCode runs immediately after you execute InsertItem. If this property is set to False, any RowInsert PeopleCode runs after you execute the Save method.

The inserted item is not added to the database until after you use the Save method.

Example

In the following example a new item (row of data) is added at the *end* of the current collection.

```
&MYDC = &MYCI.EMPL_CHECKLIST;
&COUNT = &MYDC.Count;
&ITEM = &MYDC.InsertItem(&COUNT);
&ITEM.CHECKLIST_CD = "00001";
&ITEM.RESPONSIBLE_ID = "6609";
&RSLT = &MYCI.Save();
```


Item

Syntax

`Item``(`*`number`*`)`

Description

The Item returns the item (row of data) that exists at the *number* position in the data collection executing the method. The parameter takes a number value.

ItemByKeys

Syntax

`ItemByKeys``(`*`key_values`*`)`

Description

The ItemByKeys method returns the item specified by the parameters. The number and type of keys are unique to each specific collection. Each key must be separated by a comma.

The collection reference must be the name of the Component Interface, followed by the record name. This method won't work on a collection reference (that is, &CI.RECNAME.ItemByKeys, not &MYCOLLECTION.ItemByKeys).

After you've returned an item, use the ItemNum property to determine the number of the item.

The keys must be in the *exact* order as in the Component Interface. A second level data collection also contains the keys of the parent data collection.

An easy way to determine the keys and their order in PeopleCode is to open the Component Interface in Application Designer, and use the Test Component. To determine the keys in Visual Basic, use the Object Browser.

See Chapter 12, "Component Interface Classes," ItemNum, page 643.

To see the signature for ItemByKeys:

1. Open the Component Interface in Application Designer.
2. Start the Component Interface Tester.

   Select the open Component Interface, then right-click, and select Test Component Interface from the pop-up menu.
3. Instantiate an object.

   Add data to the Get or Create keys and click Get Existing or Create New, respectively.

4. Expand the instantiated component until you find the collection in which you're interested.
5. Right-click on the collection and select ItemByKeys from the resulting pop-up menu.
6. The dialog that follows shows you the specific parameters, types, and order in which you should call ItemByKeys.

Returns

An item (row) of data from a data collection.

Example

```
Local ApiObject &MYSESSION;
Local ApiObject &CI;
Local ApiObject &CI_COLLECTION;
Local ApiObject &CI_ITEM;

&MYSESSION = %Session;
&CI = &MYSESSION.GetCompIntfc(COMPINTFC.CM_EVAL);
&CI.EMPLID = "8001";
If &CI.Get() <> 1 Then
   Exit;
End-If;

&CI_COLLECTION = &CI.CM_EVALUATIONS;
&COUNT = &CI_COLLECTION.Count;

&CI_ITEM = &CI.CM_EVALUATIONS.itembykeys("01");
&CI_ITEM.DESCR50 = "TEST";
If &CI.Save() <> 1 Then
   Exit;
End-If;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Component Interfaces, "Developing Component Interfaces," Testing Component Interfaces

Data Collection Properties

This section explains the following Data Collection properties:

- Count
- CurrentItemNumber

Count

Description

This property returns the number of data items (rows of data) in the data collection.

This property is read-only.

Example

```
&CI = &MYSESSION.GetCompIntfc(COMPINTFC.CM_EVAL_BC);
&CI.EMPLID = "8001";
&CI.Get()

&CI_COLLECTION = &CI.CM_EVALUATIONS;
&COUNT = &CI_COLLECTION.Count;
```

CurrentItemNumber

Description

If the component associated with the Component Interface is effective-dated, this property returns the item number for the current effective-dated item (row of data).

This property is read-only.

Data Item Class Property

This section discusses the ItemNum property.

ItemNum

Description

This property returns the number of the data item (row) in the collection. For example, many of the data collection methods takes a number as a parameter. Use this property to determine the item number (row number) of an item in a collection, then use that number in another method.

This property is read-only.

Example

```
Evaluate USER_ACTION

.  .  .

When = "D"
   &CI_ITEM = &CI_LVL1_NAMES.ItemByKeys(&NAME_TYPE, &NAME_PART);
   If &CI_ITEM <> Null then
      &I = &CI_LVL1_NAMES.ItemNum;
      &CI_LVL1_NAMES.DeleteItem(&I);
   End-if;
.  .  .
End-Evaluate;
```

Accessing the Structure of a Component Interface

The structure of a Component Interface can be accessed using a CompIntfPropInfoCollection object. You access a CompIntfPropInfoCollection object from a CompIntfPropInfoCollection collection. There is more than one way to instantiate a CompIntfPropInfoCollection collection.

Note. You don't have to populate a Component Interface with data before you access the structure. You can access the structure of a Component Interface immediately after you use the GetCompIntfc method with a session object. Accessing the structure of a Component Interface before you populate it with data may increase your performance.

See Chapter 12, "Component Interface Classes," GetCompIntfc, page 613.

CompIntfPropInfoCollection Collection

There are two types of CompIntfPropInfoCollection object properties: *field* properties and *collection* properties.

A *field* property maps to a specific record field. You can access structural information about the field using a CompIntfPropInfoCollection object. This information includes the name of the field, whether it's required, is it based on a prompt table, and so on.

A *collection* property is just that, a collection of properties. And before you can access a CompIntfPropInfoCollection object, you must first get a CompIntfPropInfoCollection *collection.* The following are the valid types of CompIntfPropInfoCollection collections:

- CREATEKEYS, GETKEYS and FINDKEYS

  When you create a component, you must specify the search record to be used with that component. You can also specify an alternate search record to be used when the component is accessed in Add mode. The key fields from those records make up the GETKEYS, FINDKEYS, and CREATEKEYS collections.

  Note.

  In order for a component interface to validate a key against a prompt table, both the Search Edit and List Box Item options must be selected in the record field properties for the key.

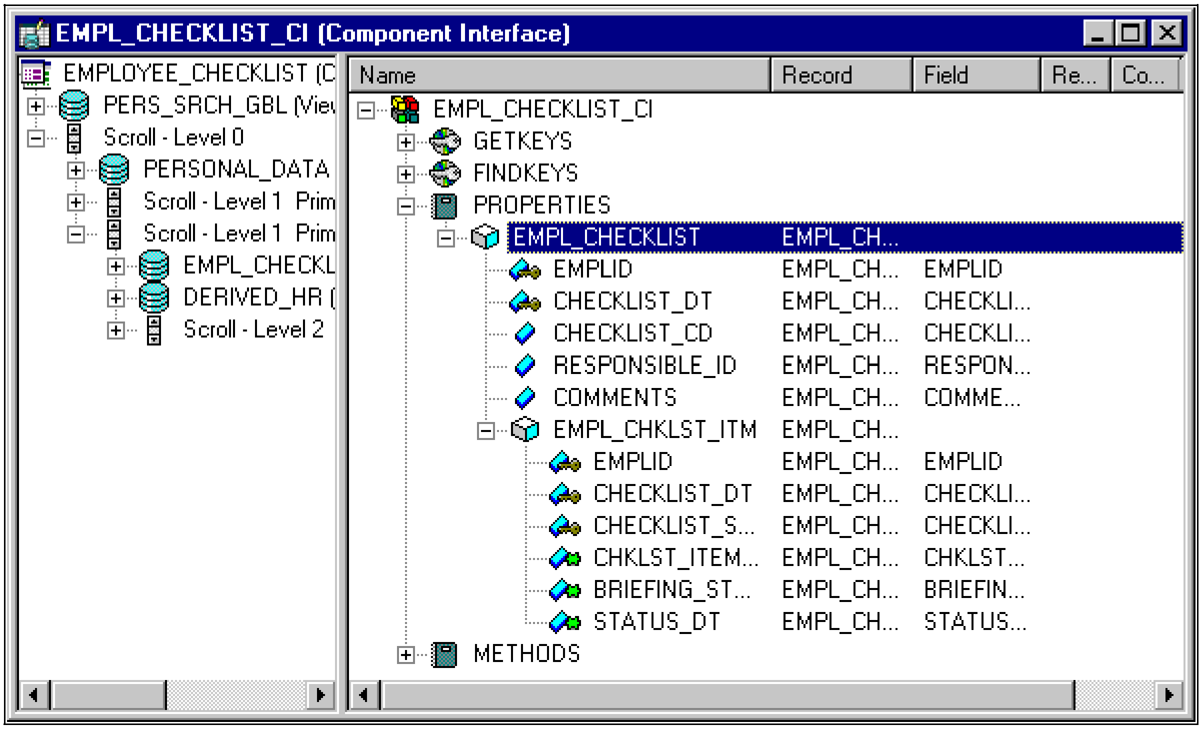
  - CREATEKEYS: A collection containing the key fields of the search record specified to be used in Add mode. Use the CreateKeyInfoCollection property to instantiate the CompIntfPropInfoCollection collection.
  - GETKEYS: A collection containing the primary required key fields from the primary search record. Use the GetKeyInfoCollection property to instantiate the CompIntfPropertInfo collection.
  - FINDKEYS: A collection containing the key fields and the alternate key fields from the primary search record. Use the FindKeyInfoCollection property to instantiate the CompIntfPropertInfo collection.

- A page scroll

  The fields associated with a page scroll are in this type of collection. This may or may not be all the fields associated with the record. Use the PropertyInfoCollection property to instantiate this kind of collection.

If the page the Component Interface is based on contains a secondary scroll, you can check the Type property to determine if the object is a CompIntfPropInfoCollection object (that is, a field), or a scroll. Then, to get the properties of the fields associated with that secondary scroll, you can use the PropertyInfoCollection property on the CompIntfPropInfoCollection object.

For example, the following Component Interface has a level zero and level one scroll.



Component Interface with secondary scroll

The level zero scroll is made up of three fields: CHECKLIST_CD, RESPONSIBLE_ID, COMMENTS, and a level one scroll, EMPL_CHKLST_ITM. The CompIntfPropInfoCollection collection for this Component Interface would have four items in it. Three would be CompIntfPropInfoCollection objects (the three fields.) The last item, EMPL_CHKLST_ITM would *not* be a valid CompIntfPropInfoCollection object. You can use the IsCollection property to verify if an item in a collection is itself a collection or a valid CompIntfPropInfoCollection object.

To access the fields in a lower level scroll, you must use the PropertyInfoCollection property first, to return a collection of those fields.

The following example loops through a Component Interface. It pulls out the names of the properties in the first three levels of a Component Interface. If the property is a nested collection, it prefixes the ancestor collection name to the property name.


```
&MYSESSION = %Session;
&CI = &MYSESSION.GetCompIntfc(COMPINTFC.VOLNEW);
&PROPINFO_0 = &CI.PropertyInfoCollection;
For &I = 1 To &PROPINFO_0.Count;
   &PROPITEM_0 = &PROPINFO_0.Item(&I);
   Warning (&PROPITEM_0.Name);
   If (&PROPITEM_0.IsCollection) Then
      &PROPINFO_1 = &PROPITEM_0.PropertyInfoCollection;
      For &J = 1 To &PROPINFO_1.Count;
         &PROPITEM_1 = &PROPINFO_1.Item(&J);
         &S1 = &PROPITEM_0.Name | "." | &PROPITEM_1.Name;
         Warning (&S1);
         If (&PROPITEM_1.IsCollection) Then
            &PROPINFO_2 = &PROPITEM_1.PropertyInfoCollection;
            For &K = 1 To &PROPINFO_2.Count;
               &PROPITEM_2 = &PROPINFO_2.Item(&K);
               &S1 = &PROPITEM_0.Name | "." | &PROPITEM_1.Name | "." | ⇒
&PROPITEM_2.Name;
               Warning (&S1);
            End-For;
         End-If
      End-For;
   End-If;
End-For;
```

CompIntfPropInfoCollection Collection Methods

This section explains the CompIntfPropInfoCollection collection methods.

First

Syntax

First()

Description

The First method returns the first CompIntfPropInfoCollection object in the CompIntfPropInfoCollection collection object executing the method.

Example

```
&CIINFO = &CIPROPCOLL.First();
```

Item

Syntax

`Item``(`*`number`*`)`

Description

The Item method returns a CompIntfPropInfoCollection object that exists at the *number* position in the CompIntfPropInfoCollection collection executing the method

If the specified item is *itself* a collection, the CompIntfPropInfoCollection object that gets returned is invalid (set to NULL). You can use the PropertyInfoCollection property to return a collection of CompIntfPropInfoCollection objects for the collection.

Example

```
&SCROLL1 = &MYCI.PropertyInfoCollection;
For &I = 1 to &SCROLL1.Count;
&PROPERTY = &SCROLL1.Item(&I);
If &PROPERTY.IsCollection Then
&SCROLL2 = &PROPERTY.PropertyInfoCollection;
/*do scroll 2 processing */
Else
/* do scroll 1 processing */
End-If;
End-For;
```

Next

Syntax

`Next``()`

Description

The Next method returns the next CompIntfPropInfoCollection object in the CompIntfPropInfoCollection collection object executing the method. You can use this method only after you have used either the First or Item methods: otherwise the system doesn't know where to start.

CompIntfPropInfoCollection Collection Properties

This section explains the Count property.

Count

Description

This property returns the number of CompIntfPropInfoCollection objects in the CompIntfPropInfoCollection collection object executing the method.

This property is read-only.

Example

```
&COUNT = &MYCIPROPINFOC.Count;
```

CompIntfPropInfoCollection Object Properties

This section explains the CompIntfPropInfoCollection Object properties.

Format

Description

This property returns the field format for the object executing the property (that is, name, phone, zip, SSN, and so on) as a number. The values are:

Value	***Description***
0	No Format
1	Name
2	Phone
3	Zip
4	SSN
5	Routine
6	Mixed Case
7	Raw Binary
8	Number only

Value	*Description*
9	SIN
10	Phone International
11	Zip International
12	Seconds
13	Microseconds
14	Custom

This property is read-only.

IsCollection

Description

This property returns True if the object executing the property is a data collection, False otherwise. If IsCollection is True, other field-oriented properties like Required, Type, Xlat, YesNo, Prompt and Format are undefined. If IsCollection is False, the object represents a field and all the previous properties are defined as described.

This property is read-only.

IsReadOnly

Description

This property returns True if the property marked Read Only in the Component Interface Definition; False otherwise.

This property is read-only.

Key

Description

This property returns True if the object executing the property is a key, False otherwise.

This property is read-only.

LabelLong

Description

This property returns the record field LongName value as a string. If there is a component override for this value, it is *not* included.

This property is read-only.

LabelShort

Description

This property returns the record field ShortName value as a string. If there is a component override for this value, it is *not* included.

This property is read-only.

Length

Description

This property returns the length of the field property as a number. If the property is a collection, this property returns a zero.

The length of the field is calculated by converting it to a string then getting the length of the string. This means that *everything* in the field is counted as a character, including spaces, date time separation characters, decimal points, sign indicators, and so on.

For example, the following string has 10 characters:

```
01/01/2001
```

The following string has five characters:

```
10.10
```

The following string has six characters:

```
-10.10
```

This property is read-only.

Name

Description

This property returns the name of the object executing the property as a string.

This property is read-only.

Prompt

Description

This property returns True if the object executing the property is associated with a prompt table, False otherwise.

This property is read-only.

PropertyInfoCollection

Description

This property returns another CompIntfPropInfoCollection collection if the object executing the property is a collection.

This property is read-only.

Example

```
&SCROLL1 = &MYCI.PropertyInfoCollection;
For &I = 1 to &SCROLL1.Count;
   &PROPERTY = &SCROLL1.Item(&I);
   If &PROPERTY.IsCollection Then
   &SCROLL2 = &PROPERTY.PropertyInfoCollection;
   /*do scroll 2 processing */
   Else
/* do scroll 1 processing */
   End-If;
End-For;
```

Required

Description

This property returns True if the object executing the property is a required property, False otherwise.

This property is read-only.

Type

Description

This property returns the field type, as a number, of the object. The values are:

Value	***Description***
0	Character
1	Long Character
2	Number
3	Signed Number
4	Date
5	Time
6	DateTime
7	SubRecord (Not supported with Component Interfaces)
8	Image (Limited support with Component Interfaces)
9	ImageReference (Not supported with Component Interfaces)

This property is read-only.

Xlat

Description

This property returns True if the object executing the property is associated with an XLAT table, False otherwise.

This property is read-only.

YesNo

Description

This property returns True if the object executing the property is associated with the Yes/No table, False otherwise.

This property is read-only.

Component Interface Examples

The following are examples of some of the most usual actions you're likely to perform using a Component Interface.

Create a New Instance of Data Example

The following is an example of how to create a new instance of a Component Interface.

To create a new instance of data:

In this example, you are creating a new instance of data for the EXPRESS Component Interface, which is based on the EXPRESS_ISSUE_INV component. The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;


&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EXPRESS);
&MYCI.BUSINESS_UNIT = "H01B";
&MYCI.INTERNAL_FLG = "Y";
&MYCI.ORDER_NO = "NEXT&rsquo;;
&MYCI.Create();
&MYCI.CUSTOMER = "John&rsquo;s Chicken Shack";
&MYCI.LOCATION = "H10B6987";
.
.
.
If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```


1. Get a session object.

   Before you can get a Component Interface, you have to get a session object. The session controls access to the Component Interface, provides error tracing, enables you to set the runtime environment, and so on.

   ```
   &MYSESSION = %Session;
   ```

2. Get a Component Interface.

   Use the GetCompIntfc method with a session object to get the Component Interface. You must specify a Component Interface definition that has already been created. You receive a runtime error if you specify a Component Interface that doesn't exist.

   ```
   &MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EXPRESS);
   ```

   After you execute the GetCompIntfc method, you have only the *structure* of the Component Interface. You haven't populated the Component Interface with data yet.

3. Set the CREATEKEYS.

   These key values are required to open a new instance of the data. If the values you specify aren't unique, that is, if an instance of the data already exists in the database with those key values, you receive a runtime error.

   ```
   &MYCI.BUSINESS_UNIT = "H01B";
   &MYCI.INTERNAL_FLG = "Y";
   &MYCI.ORDER_NO = "NEXT&rsquo;;
   ```

4. Create the instance of data for the Component Interface.

   After you set the key values, you must use the Create method to populate the Component Interface with the key values you set.

   ```
   &MYCI.Create();
   ```

   This creates an instance of this data. However, it hasn't been saved to the database. You must use the Save method before the instance of data is committed to the database.

5. Set the rest of the fields.

   Assign values to the other fields.

   ```
   &MYCI.CUSTOMER = "John&rsquo;s Chicken Shack";
   &MYCI.LOCATION = "H10B6987";
   .
   .
   .
   ```

   If you have specified InteractiveMode as True, every time you assign a value or use the InsertItem or DeleteItem methods, any PeopleCode programs associated with that field (either with record field or the component record field) fires (FieldEdit, FieldChange, RowInsert, and so on.)

6. Save the data.

   When you execute the Save method, the new instance of the data is saved to the database.

   ```
   If NOT(&MYCI.Save()) Then
   ```

   The Save method returns a Boolean value: True if the save was successful, False otherwise. You can use this value to do error checking.

   The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with SaveEdit, SavePreChange, WorkFlow, and so on.) are executed. If you didn't specify the Component Interface to run in interactive mode, FieldEdit, FieldChange, and so on, also run at this time for all fields that had values set.

   Note. If you're running a Component Interface from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

   ---

7. Check Errors.

   You can check if there were any errors using the PSMessages property on the session object.

   ```
   If NOT(&MYCI.Save()) Then
      /* save didn&rsquo;t complete */
      &COLL = &MYSESSION.PSMessages;
      For &I = 1 to &COLL.Count
         &ERROR = &COLL.Item(&I);
         &TEXT = &ERROR.Text;
         /* do error processing */
      End-For;
      &COLL.DeleteAll();
   End-if;
   ```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, delete it from the PSMessages collection.

   The Text property of the PSMessage returns the text of the error message. At the end of this text is a contextual string that contains the name of the field that generated the error. The contextual string has the following syntax:

   ```
   {BusinessComponentName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
   (Row)]]].PropertyName}
   ```

   For example, if you specified the incorrect format for a date field of the Component Interface named ABS_HIST, the Text property would contain the following string:

   ```
   Invalid Date {ABS_HIST.BEGIN_DT} (90), (1)
   ```

   The contextual string (by itself) is available using the Source property of the PSMessage.

   Note. If you've called your Component Interface from an Application Engine program, all errors are also logged in the Application Engine error log tables.

   See Chapter 38, "Session Class," Error Handling, page 2180 and Chapter 38, "Session Class," Source, page 2196.

Getting an Existing Instance of Data Example

The following is an example of how to get an existing instance of a Component Interface.

In this example, you are getting an existing instance of data for the EMPL_CHKLST_BC Component Interface, which is based on the EMPLOYEE_CHECKLIST component. The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHKLST_BC);
&MYCI.EMPLID= "8001";
&MYCI.Get();

/* Get checklist Code */

&CHECKLIST_CD = &MYCI.CHECKLIST_CD;

/* Set Effective date */

&MYCI.EFFDT = "05-01-1990";
.
.
.
If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

1. Get a session object.

   Before you can get a Component Interface, you have to get a session object. The session controls access to the Component Interface, provides error tracing, enables you to set the runtime environment, and so on.

   ```
   &MYSESSION = %Session;
   ```

2. Get a Component Interface.

   Use the GetCompIntfc method with a session object to get the Component Interface. You must specify a Component Interface definition that has already been created. You receive a runtime error if you specify a Component Interface that doesn't exist.

   ```
   &MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHKLST_BC);
   ```

   After you execute the GetCompIntfc method, you have only the *structure* of the Component Interface. You haven't populated the Component Interface with data yet.

3. Set the GETKEYS.

   These are the key values required to return a unique instance of existing data. If the keys you specify allow for more than one instance of the data to be returned, or if no instance of the data matching the key values is found, you receive a runtime error.

   ```
   &MYCI.EMPLID = "8001";
   ```

4. Get the instance of data for the Component Interface.

   After you set the key values, you have to use the Get method.

   ```
   &MYCI.Get();
   ```

   This populates the Component Interface with data, based on the key values you set.

5. Get field values or set field values.

   At this point, you can either get values or set values.

   ```
   &CHECKLIST_CD = &MYCI.CHECKLIST_CD;
   ```

   ```
   /* OR */
   &MYCI.EFFDT = "05-01-1990";
   ```

   If you have specified InteractiveMode as True, every time you assign a value, any PeopleCode programs associated with that field (either with record field or the component record field) fires (FieldEdit, FieldChange, and so on.)

6. Save or Cancel the Component Interface, as appropriate.

   If you've changed values and you want to save your changes to the database, you must use the Save method.

   ```
   If NOT(&MYCI.Save()) Then
   ```

   The Save method returns a Boolean value: True if the save was successful, False otherwise. Use this value to do error checking.

   The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with SaveEdit, SavePreChange, WorkFlow, and so on.) are executed. If you didn't specify the Component Interface to run in interactive mode, FieldEdit, FieldChange, and so on, also run at this time.

   Note. If you're running a Component Interface from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

   If you don't want to save any changes to the data, use the Cancel method. The Component Interface is reset to the state it was in just after you used the GetCompIntfc method.

   ```
   &MYCI.Cancel();
   ```

   This is similar to a user pressing ESC while in a component, and choosing to not save any of their changes.

7. Check Errors.

   You can check if there were any errors using the PSMessages property on the session object.

```
If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, delete it from the PSMessages collection.

   The Text property of the PSMessage returns the text of the error message. At the end of this text is a contextual string that contains the name of the field that generated the error. The contextual string has the following syntax:

```
{ComponentInterfaceName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
(Row)]]].PropertyName}
```

   For example, if you specified the incorrect format for a date field of the Component Interface named ABS_HIST, the Text property would contain the following string:

```
Invalid Date {ABS_HIST.BEGIN_DT} (90), (1)
```

   The contextual string (by itself) is available using the Source property of the PSMessage.

   Note. If you've called your Component Interface from an Application Engine program, all errors are also logged in the Application Engine error log tables.

   See Chapter 38, "Session Class," Error Handling, page 2180 and Chapter 38, "Session Class," Source, page 2196.

Retrieving a List of Instance of Data Example

The following is an example of how to retrieve a list of instances of data.

To retrieve a list of instances of data:

In this example, you are getting a list of existing instances of data for the EMPL_CHKLST_CI Component Interface, which is based on the EMPLOYEE_CHECKLIST component. The following is the complete code sample: the steps break it up and explain each line.


```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;
Local ApiObject &MYNEWCI;


&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHKLST_CI);
&MYCI.EMPLID= "8";
&MYCI.LAST_NAME_SRCH = "S";
&MYLIST = &MYCI.Find();
For &I = 1 to &MYLIST.Count;

   /* note: do not reuse the CI used to create the list, or the list will be⇒
 destroyed */

&MYNEWCI = &MYLIST.Item(&I);

/* CI from list still must be instantiated to use it */

&MYNEWCI.Get();

/* do some processing */

End-For;
```

1. Get a session object.

   Before you can get a Component Interface, you have to get a session object. The session controls access to the Component Interface, provides error tracing, enables you to set the runtime environment, and so on.

   ```
   &MYSESSION = %Session;
   ```

2. Get a Component Interface.

   Use the GetCompIntfc method with a session object to get the Component Interface. You must specify a Component Interface definition that has already been created. You receive a runtime error if you specify a Component Interface that doesn't exist.

   ```
   &MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHKLST_CI);
   ```

   After you execute the GetCompIntfc method, you have only the *structure* of the Component Interface. You haven't populated the Component Interface with data yet.

3. Set the FINDKEYS values.

   These can be alternate key values or partial key values. If no instance of the data matching the key values is found, you receive a runtime error.

   ```
   &MYCI.EMPLID = "8";
   &MYCI.LAST_NAME_SRCH = "S";
   ```

4. Get a list of data instances for the Component Interface.

   After you set the alternate or partial key values, use the Find method to return a list of data instances for the Component Interface.

   ```
   &MYLIST = &MYCI.Find();
   ```

   Note. The Find method can be executed only at level zero in a Component Interface.

   ---

5. Select an instance of the data.

   To select an instance of the data, you can use any of the following standard data collection methods:

   - First
   - Item
   - Next

For example, you could loop through every instance of the data to do some processing:

```
For &I = 1 to &MYLIST.Count;
   /* note: do not reuse the Component Interface used to */
   /* create the list here, or the list will be destroyed */
&MYNEWCI = &MYLIST.Item(&I);
/* CI from list must still be instantiated to use it */
&MYNEWCI.Get();
/* do some processing */
End-For;
```

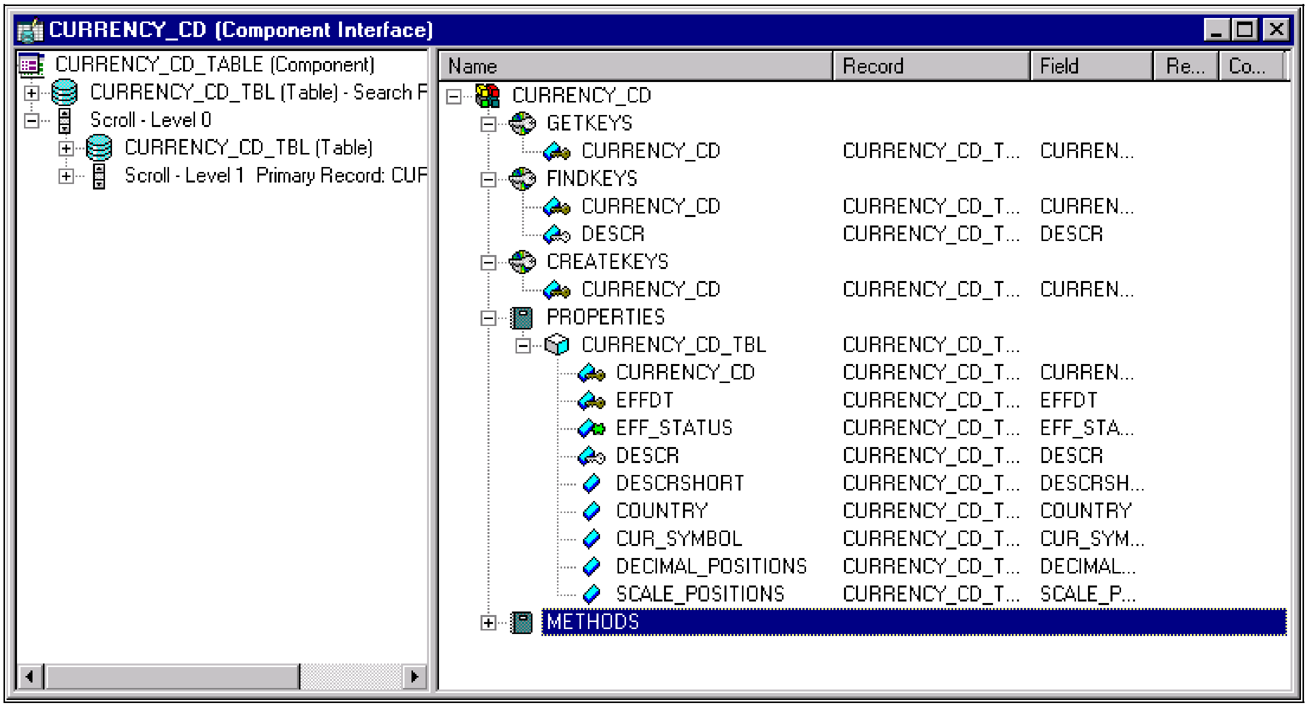
After you have a specific instance of the data, you can get values, set values, and so on.

See Chapter 12, "Component Interface Classes," Data Collection Methods, page 636.

Inserting Effective-Dated Data Example

You can rename a collection in a Component Interface. For example, suppose you had the same record at level zero and at level one. You may want to rename the level one data collection to reflect this. The data in a data collection is associated with the primary database record of a scroll, *not* with the name you supply.

Here is an example of using a Component Interface that has the same record at level zero and level one. The Component Interface is based on the CURRENCY_CD_TBL component.



Example of CI with same record at level zero and level one

The following code example is based on this Component Interface and does the following:

1. Gets an existing Component Interface.
2. Finds the current effective-dated item index.

3. Inserts a new row before the current effective-dated item using the InsertItem method.

```
Local ApiObject &Session;
Local ApiObject &CURRENCY_CD;
Local ApiObject &CURRENCY_CD_TBLCol;
Local ApiObject &CURRENCY_CD_TBLItm;
Local ApiObject &PSMessages;
Local string &ErrorText, &ErrorType;
Local number &ErrorCount;
Local Boolean &Error;

Function CheckErrorCodes()

   &PSMessages = &Session.PSMessages;
   &ErrorCount = &PSMessages.Count;
   For &i = 1 To &ErrorCount
      &ErrorText = &PSMessages.Item(&i).Text;
      &ErrorType = &PSMessages.Item(&i).Type;
   End-For;

End-Function;

/* Initialize Flags */
&Error = False;

&Session = %Session;

If &Session <> Null Then

   CheckErrorCodes();
   /* Application Specific Error Processing */

Else

   &CURRENCY_CD = &Session.GetCompIntfc(CompIntfc.CURRENCY_CD);

   If &CURRENCY_CD = Null Then

      CheckErrorCodes();
      /* Application Specific Error Processing */

   Else

      /* Set Component Interface Get Keys */
      &CURRENCY_CD.CURRENCY_CD = "USD";

      If Not &CURRENCY_CD.Get() Then

         CheckErrorCodes();
         /* Application Specific Error Processing */
         &Error = True;

      End-If;

      If Not &Error Then

         &CURRENCY_CD_TBLCol = &CURRENCY_CD.CURRENCY_CD_TBL;
         &CURRENCY_CD_TBLItm = &CURRENCY_CD_TBLCol.InsertItem(&CURRENCY_CD_⇒
TBLCol.CurrentItemNum());
         &CURRENCY_CD_TBLItm.EFFDT = %Date;
         &CURRENCY_CD_TBLItm.EFF_STATUS = "A";
         &CURRENCY_CD_TBLItm.DESCR = "NewCurrencyCode";
         &CURRENCY_CD_TBLItm.DESCRSHORT = "New";
```

```
            &CURRENCY_CD_TBLItm.COUNTRY = "USA";
            &CURRENCY_CD_TBLItm.CUR_SYMBOL = "?";
            &CURRENCY_CD_TBLItm.DECIMAL_POSITIONS = 4;
            &CURRENCY_CD_TBLItm.SCALE_POSITIONS = 0;

            /* Save Instance of Component Interface */
            If Not &CURRENCY_CD.Save() Then

               CheckErrorCodes();
               /* Application Specific Error Processing */

            End-If;

         End-If;
         /* End: Set Component Interface Properties */

         /* Cancel Instance of Component Interface */
         &CURRENCY_CD.Cancel();

      End-If;

End-If;
```

Inserting Effective-Dated Data Example Using Visual Basic

Here's a code example in Visual Basic that does the same thing as the previous code example.


```
Private Sub CURRENCY_CD()

        On Error GoTo eMessage

        '***** Set Object References *****
        Dim oCISession As Object
        Dim oCURRENCY_CD As Object
        Dim oCURRENCY_CD_TBL As Object
        Dim oCURRENCY_CD_TBLItem As Object

        '***** Set Connect Parameters *****
        strAppSeverPath = "//PSOFT101999:9000"
        strOperatorID = "PTDMO"
        strPassword = "PTDMO"

        '***** Create PeopleSoft Session Object *****
        Set oCISession = CreateObject("PeopleSoft.Session")

        '***** Connect to the App Sever *****
        oCISession.Connect 1, strAppSeverPath, strOperatorID, strPassword, 0

        '***** Get the Component *****
        Set oCURRENCY_CD = oCISession.GetCompIntfc("CURRENCY_CD")

        '***** Set the Component Interface Mode *****
        oCURRENCY_CD.InteractiveMode = False
        oCURRENCY_CD.GetHistoryItems = True

        '***** Set Component Get/Create Keys *****
        oCURRENCY_CD.CURRENCY_CD = "USD"

        '***** Execute Create *****
        oCURRENCY_CD.Get

        'Set CURRENCY_CD_TBL Collection Field Properties --
        'Parent: PS_ROOT Collection
        Set oCURRENCY_CD_TBL = oCURRENCY_CD.CURRENCY_CD_TBL
        Set oCURRENCY_CD_TBLItem = oCURRENCY_CD_TBL.InsertItem(oCURRENCY_CD_⇒
TBL.CurrentItemNum())

        oCURRENCY_CD_TBLItem.EFFDT = Date
        oCURRENCY_CD_TBLItem.EFF_STATUS = "A"
        oCURRENCY_CD_TBLItem.DESCR = "NewCurrencyCode"
        oCURRENCY_CD_TBLItem.DESCRSHORT = "New"
        oCURRENCY_CD_TBLItem.COUNTRY = "USA"
        oCURRENCY_CD_TBLItem.CUR_SYMBOL = "?"
        oCURRENCY_CD_TBLItem.DECIMAL_POSITIONS = 4
        oCURRENCY_CD_TBLItem.SCALE_POSITIONS = 0

        '***** Save Component Interface *****
        oCURRENCY_CD.Save
        oCURRENCY_CD.Cancel

        Exit Sub

eMessage:
        '***** Display VB Runtime Errors *****
        MsgBox Err.Description

        '***** Display PeopleSoft Error Messages *****
        If oCISession.PSMessages.Count > 0 Then
                For i = 1 To oCISession.PSMessages.Count
                        MsgBox oCISession.PSMessages.Item(i).Text
                Next i
```



```
        End If

End Sub

Sub MAIN()
    CURRENCY_CD
End Sub
```

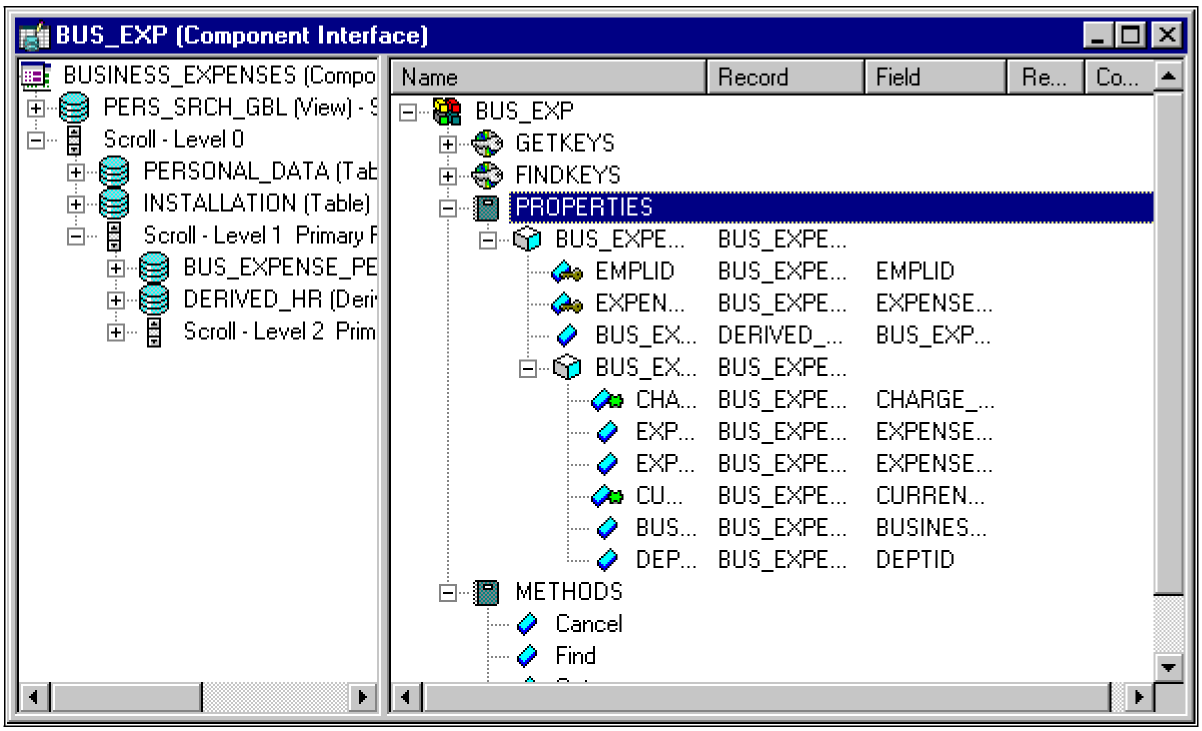
Inserting or Deleting a Row of Data Example

The CopyRowset and CopyRowsetDelta methods use the primary database name of a scroll, not the name you may give a collection.

A data collection represents a row of data. You often insert or delete a row of data.

To insert or delete a row of data:

In this example, you are getting a existing instance of data for the BUS_EXP Component Interface, which is based on the BUSINESS_EXPENSES component, then inserting (or deleting) a row of data in the second level scroll.



BUS_EXP Component Interface definition

The following is the complete code sample: the steps explain each line.


```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.BUS_EXP);
&MYCI.EMPLID= "8001";
&MYCI.Get();
&MYLEVEL1 = &MYCI.BUS_EXPENSE_PER;
/* get appropriate item in lvl1 collection */
For &I = 1 to &MYLEVEL1.Count
&ITEM = &MYLEVEL1.Item(&I);
&MYLEVEL2 = &ITEM.BUS_EXPENSE_DTL;
&COUNT = &MYLEVEL2.Count
/* get appropriate item in lvl2 collection */
For &J = 1 to &COUNT
&LVL2ITEM = &MYLEVEL2.Item(&J);
&CIDATE = &LVL2ITEM.CHARGE_DT;
If &CIDATE <> %Date Then
      /* insert row */
&NEWITEM = &MYLEVEL2.InsertItem(&COUNT);
   /* set values for &NEWITEM */
Else
      /* delete last row */
   &MYLEVEL2.DeleteItem(&COUNT);
End-If;
   End-For;
End-For;

If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

1. Get a session object.

   Before you can get a Component Interface, you have to get a session object. The session controls access to the Component Interface, provides error tracing, enables you to set the runtime environment, and so on.

   ```
   &MYSESSION = %Session;
   ```

2. Get a Component Interface.

   Use the GetCompIntfc method with a session object to get the Component Interface. You must specify a Component Interface definition that has already been created. You receive a runtime error if you specify a Component Interface that doesn't exist.

   ```
   &MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.BUS_EXP);
   ```

   After you execute the GetCompIntfc method, you have only the *structure* of the Component Interface. You haven't populated the Component Interface with data yet.

3. Set the GETKEYS.

   These are the key values required to return a unique instance of existing data. If the keys you specify allow for more than one instance of the data to be returned, or if no instance of the data matching the key values is found, you receive a runtime error.

   ```
   &MYCI.EMPLID = "8001";
   ```

4. Get the instance of data for the Component Interface.

   After you set the key values, you must use the Get method.

   ```
   &MYCI.Get();
   ```

   This populates the Component Interface with data, based on the key values you set.

5. Get the level one scroll.

   The name of the scroll can be treated like a property. It returns a data collection that contains all the level one data.

   ```
   &MYLEVEL1 = &MYCI.BUS_EXPENSE_PER
   ```

6. Get the appropriate item in the level one data collection.

   Remember, scroll data is hierarchical. You must get the appropriate level one row before you can access the level two data.

   ```
   For &I = 1 to &MYLEVEL1.Count
   &ITEM = &MYLEVEL1.Item(&I);
   ```

7. Get the level two scroll.

   This is done the same way as you accessed the level one scroll, by using the scroll name as a property.

   ```
   &MYLEVEL2 = &ITEM.BUS_EXPENSE_DTL;
   ```

8. Get the appropriate item in the level two data collection.

   A data collection is made up of a series of items (rows of data.) You have to access the appropriate item (row) in this level also.

   ```
   &COUNT = &MYLEVEL2.Count
   /* get appropriate item in lvl2 collection */
   For &J = 1 to &COUNT
   &LVL2ITEM = &MYLEVEL2.Item(&J);
   ```


9. Insert or delete a row of data.

   You can insert or delete a row of data from a data collection. The following example finds the last item (row of data) in the second level scroll. If it matches the value of %Date, the last item is deleted. If it doesn't match, a new row is inserted.

```
&CIDATE = &LVL2ITEM.CHARGE_DT;
If &CIDATE <> %Date Then
   /* insert row */
&NEWITEM = &MYLEVEL2.InsertItem(&COUNT);
/* set values for &NEWITEM */
Else
   /* delete last row */
&MYLEVEL2.DeleteItem(&COUNT);
End-If;
```

10. Save the data.

    When you execute the Save method, the new instance of the data is saved to the database.

```
If NOT(&MYCI.Save()) Then
```

    The Save method returns a Boolean value: True if the save was successful, False otherwise. Use this value to do error checking.

    The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with SaveEdit, SavePreChange, WorkFlow, and so on) are executed. If you did not specify the Component Interface to run in interactive mode, FieldEdit, FieldChange, and so on, also run at this time.

    Note. If you're running a Component Interface from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

11. Check Errors.

    You can check if there were any errors using the PSMessages property on the session object.

```
If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &SESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

    If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, delete it from the PSMessages collection.

    The Text property of the PSMessage returns the text of the error message. At the end of this text is a contextual string that contains the name of the field that generated the error. The contextual string has the following syntax:

```
{BusinessComponentName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
(Row)]]].PropertyName}
```

    For example, if you specified the incorrect format for a date field of the Component Interface named ABS_HIST, the Text property contains the following string:

```
Invalid Date {ABS_HIST.BEGIN_DT} (90), (1)
```

    The contextual string (by itself) is available using the Source property of the PSMessage.

    Note. If you've called the Component Interface from an Application Engine program, all errors are also logged in the Application Engine error log tables.

    See Chapter 38, "Session Class," Error Handling, page 2180 and Chapter 38, "Session Class," Source, page 2196.

Chapter 13

Connected Query Classes

This chapter provides an overview of the connected query classes and discusses:

- Importing connected query classes.
- CONQRSMGR class.
- CONQRSPROPERTY class.
- CONQRS_CONST class.
- QUERYITEMPROMPT class.
- SCHED_INFO class.
- SEC_PROFILE class.
- UTILITY class.

Understanding the Connected Query Classes

Connected query is a PeopleTools managed object that supports typical managed object functionality such as:

- Creating a new object.
- Editing an existing object.
- Deleting an object.
- Copying an object.
- Renaming an object.[1]
- Caching an object.
- Adding an object to a project.
- Using project compare functionality.
- Copying to or restoring from a file.
- Upgrading to the database.

[1] Renaming a connected query object is supported on a managed object level, but is not exposed as a function in PeopleCode.

A connected query object is exposed to the report developer through the public methods and properties of the PT_CONQRS application package. The remainder of this chapter provides reference on those classes, methods, and properties.

Connected queries are described in more detail in the PeopleSoft Query PeopleBook.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "Using Connected Query"

Importing Connected Query Classes

The connected query classes are application classes, not built-in classes, like Rowset, Field, Record, and so on. Before you can use these classes in your PeopleCode program, you must import them into your program.

An import statement either names a particular application class or imports all the classes in a package.

Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are not made available.

CONQRSMGR Class

This section provides an overview of the CONQRSMGR class and discusses:

- CONQRSMGR class constructor.
- CONQRSMGR class methods.
- CONQRSMGR class properties.

CONQRSMGR Class Constructor

This section presents the constructor for the CONQRSMGR class.

CONQRSMGR

Example

```
import PT_CONQRS:CONQRSMGR;

Local PT_CONQRS:CONQRSMGR &objConQrsInst = create PT_CONQRS:CONQRSMGR(&sOprId, ⇒
&sCONQRSNAME);
```


Note. The first parameter is not required; an empty string can be specified instead.

See Also

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

CONQRSMGR Class Methods

In this section, the CONQRSMGR class methods are presented in alphabetical order.

CleanOutputFile

Syntax

`CleanOutputFile``()`

Description

Use this method to delete the connected query output file.

Note. This method is valid only for a connected query scheduled for running to a file**—**that is, when the run mode is equal to RunMode_Sched_File.

Parameters

None.

Returns

None.

Close

Syntax

`Close``()`

Description

Use this method to clean up connected query resources.

Parameters

None.

Returns

None.

CopyDefn

Syntax

CopyDefn*(target_UserID, target_ID)*

Description

Use this method to copy the connected query definition to a new ID. The user's query access security is checked prior to the copy.

Parameters

Parameter	***Description***
target_UserID	Specifies the user ID of the owner of the new connected query as a string. An empty string indicates that the new connected query is public.
target_ID	Specifies the ID for the new connected query as a string.

Returns

A Boolean value: True if the copy is successful, False otherwise.

DeleteDefn

Syntax

DeleteDefn()

Description

Use this method to delete the current connected query definition.

Parameters

None.

Returns

A Boolean value: True if the delete was successful, False otherwise.

ExistsByName

Syntax

`ExistsByName`*`(ObjName)`*

Description

Use this method to for the existence of a connected query definition based on the specified definition name.

Parameters

Parameter	***Description***
ObjName	Specifies the connected query definition name as a string.

Returns

A Boolean value: True if connected query definition exists, False otherwise.

GetSampleXMLString

Syntax

`GetSampleXMLString`*`(RowNum)`*

Description

Use this method to generate and return a sample XML string. Use this string with XML Publisher data sources based on a connected query.

Parameters

Parameter	*Description*
RowNum	Sets number of rows of sample data to generate for each member query.

Returns

A string representing the sample XML data.

Open

Syntax

Open*(IsNew)*

Description

Use this method to open a new or existing connected query definition.

This method should be called after a connected query object constructor. It initializes a new or existed connected query object.

Parameters

Parameter	*Description*
IsNew	Specifies as a Boolean value whether the connected query object represents a new connected query definition (True), or an existing definition (False).

Returns

A Boolean value: True if the open was successful, False otherwise.

ResetQueriesPrompt

Syntax

ResetQueriesPrompt()

Description

Use this method after a user clicks Cancel in a query prompt dialog.

Parameters

None.

Returns

None.

Run

Syntax

Run*(Prompts,runProcessInfo)*

Description

Use this method to run the connected query. The Run method is valid for the following run modes only:

- RunMode_Prev **—** Referred to as *preview mode.*
- RunMode_Sched_Web or RunMode_Sched_File **—** Referred to as *scheduled mode.*

If an error results, the Run method populates an error string with error details.

Preview Mode

In preview mode, the connected query is run immediately on the application sever. The output data is displayed in a separate browser window. In preview mode, the *Prompts* parameter is required; the *runProcessInfo* parameter *must* be set to Null. The *Prompts* parameter should be retrieved using the QueriesPromptsArray property. If this array length is greater than 0, this array must be populated.

Note. If both parameters are not null, the scheduling information is used to run the connected query by Process Scheduler.

Scheduled Mode

In scheduled mode, the connected query is run by Process Scheduler. In the case of RunMode_Sched_File, the output data is written to an XML file. In the case of RunMode_Sched_Web, the output data is accessible via a hyperlink in Process Monitor. In scheduled mode, the *runProcessInfo* parameter is required; the *Prompts* parameter should be set to Null.

Note. If both parameters are not null, the scheduling information is used to run the connected query by Process Scheduler.

Parameters

Parameter	***Description***
Prompts	Specifies the prompts for execution of the queries as an array of QUERYITEMPROMPT objects. This parameter should be set to null when running in scheduled mode.
runProcessInfo	Specifies the scheduling information for running the queries as a SCHED_INFO object. This parameter *must* be null when running in preview mode.

Returns

A Boolean value: True if the connected query was run successfully, False otherwise.

See Also

Chapter 13, "Connected Query Classes," RunToWindow, page 678; Chapter 13, "Connected Query Classes," RunToXMLFormattedString, page 679; Chapter 13, "Connected Query Classes," QueriesPromptsArray, page 691; Chapter 13, "Connected Query Classes," RunMode, page 693; Chapter 13, "Connected Query Classes," QUERYITEMPROMPT Class, page 705; Chapter 13, "Connected Query Classes," SCHED_INFO Class, page 706; Chapter 13, "Connected Query Classes," Running a Connected Query in Scheduled Mode, page 713 and Chapter 13, "Connected Query Classes," Running a Connected Query in Preview Mode or Background Mode, page 715

RunToWindow

Syntax

```
RunToWindow()
```


Description

Use this method to run a connected query in *background mode.* Similar to preview mode, running a connected query in background mode results in the output data being displayed in a separate browser window. However, in background mode, the connected query is run through Process Scheduler with a REN server. Running a connected query in this manner prevents blocking on the application server. While the report file is generated, the user can continue working with the PeopleSoft application and the application server environment.

To specify the scheduling information, prior to invoking the RunToWindow method, you must invoke the SetRunControlData method as follows:

```
&res = &objConQrsInst.SetRunControlData(&objConQrsInst.Const.⇒
RunCntlId_Auto, False);
```

The RunToWindow method is valid for the RunMode_Window run mode only.

Parameters

None.

Returns

The process ID as a number.

See Also

Chapter 13, "Connected Query Classes," Run, page 677; Chapter 13, "Connected Query Classes," RunToXMLFormattedString, page 679; Chapter 13, "Connected Query Classes," RunMode, page 693 and Chapter 13, "Connected Query Classes," Running a Connected Query in Preview Mode or Background Mode, page 715

RunToXMLFormattedString

Syntax

`RunToXMLFormattedString`*`(Prompts)`*

Description

Use this method to run the connected query immediately on the application server with output to an XML formatted string instead of an XML file.

The RunToXMLFormattedString method is valid for the RunMode_XMLFormattedString run mode only.

Parameters

Parameter	*Description*
Prompts	Specifies the prompts for execution of the queries as an array of QUERYITEMPROMPT objects. This parameter should be retrieved using the QueriesPromptsArray property. If this array length is greater than 0, this array must be populated.

Returns

The formatted XML as a string.

See Also

Chapter 13, "Connected Query Classes," Run, page 677; Chapter 13, "Connected Query Classes," RunToWindow, page 678; Chapter 13, "Connected Query Classes," QueriesPromptsArray, page 691; Chapter 13, "Connected Query Classes," RunMode, page 693 and Chapter 13, "Connected Query Classes," QUERYITEMPROMPT Class, page 705

Save

Syntax

```
Save()
```

Description

Use this method to save the connected query object to the database as a connected query definition. If the connected query does not pass validation, it is saved with an inactive status.

Parameters

None.

Returns

A Boolean value: True if the save was successful, False otherwise.

SaveRunControlParms

Syntax

SaveRunControlParms*(runCntlID)*

Description

Use this method to save the in-memory process request parameter array to the database. This method is also called internally from the SetRunControlData method when *IsChangeDB* is true.

Parameters

Parameter	***Description***
runCntlID	Specifies the run control ID as a string.

Returns

A Boolean value: True if the save was successful, False otherwise.

See Also

Chapter 13, "Connected Query Classes," SetRunControlData, page 681

SetRunControlData

Syntax

SetRunControlData*(runCntlID, IsNewCtrl, IsChangeDB)*

Description

Use this as an interactive method to prompt the user if member queries have prompts. This method saves prompt data to the database using the PSCONQRSRUNPRM or PSCONQRSRUNPRMX records if the *IsChangeDB* parameter is passed as true. Call this method prior to scheduling a connected query to run in scheduled mode.

Parameters

Parameter	***Description***
runCntlID	Specifies the run control ID as a string.
IsNewCtrl	Specifies as a Boolean value whether the run control is new.
IsChangeDB	Specifies as a Boolean value whether to write the run control information to the database.

Returns

A Boolean value: True if the run control parameters were set successfully, False otherwise.

Validate

Syntax

`Validate``()`

Description

Use this method to validate the following:

- The existence of each query used in the connected query.
- Whether mapped fields used in the connected query object are still part of the QueryOutputFields collection for each query.
- The user security access for each query.

If validation fails, Validate populates an error string with the error details.

Parameters

None.

Returns

A Boolean value: True if the connected query passes all the validation checks, False otherwise.

See Also

Chapter 33, "Query Classes," QueryOutputFields, page 1971

ValidateIfFieldsMapped

Syntax

`ValidateIfFieldsMapped``()`

Description

Use this method to check whether all queries have mapped fields. Use this method to confirm that it is the report developer's intention to have unmapped fields in the queries.

If validation fails, ValidateIfFieldsMapped populates an error string with the names of the unlinked queries.

Parameters

None.

Returns

A Boolean value: True if all the queries have mapped fields, False otherwise.

ValidateRunControlParms

Syntax

`ValidateRunControlParms``(`*`runCntlID`*`)`

Description

Use this method to determine whether sufficient populated records exist in the or PSCONQRSRUNPRMX tables for the specified run control ID. ValidateRunControlParms compares the number of entries in the parameter tables with the number of prompts required for the connected query.

If validation fails, ValidateRunControlParms populates an error string with the error details.

Parameters

Parameter	*Description*
runCntlID	Specifies the run control ID as a string.

Returns

A Boolean value: True if the validation is successful, False otherwise.

CONQRSMGR Class Properties

In this section, the CONQRSMGR class properties are presented in alphabetical order.

Const

Description

This property returns a CONQRS_CONST object.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," CONQRS_CONST Class, page 699

Description

Description

Use this property to set or return a description for the connected query as a string.

This property is read-write.

Details

Description

Use this property to set or return a long description for the connected query as a string.

This property is read-write.

ErrString

Description

This property returns the error message as a string if an error occurs during processing.

This property is read-only.

ExecutionLog

Description

Use this property to return a Boolean value indicating whether to perform execution logging while executing the connected query.

This property is read-only.

Note. While this property is read-only, it can be set through the PropertyArray property.

The default value is False.

See Also

Chapter 13, "Connected Query Classes," IsDebugMode, page 686 and Chapter 13, "Connected Query Classes," PropertyArray, page 689

IgnoreRelFldOutput

Description

Use this property to return a Boolean value indicating whether to create XML output without related field nodes.

This property is read-only.

Note. While this property is read-only, it can be set through the PropertyArray property.

The default value is False.

See Also

Chapter 13, "Connected Query Classes," PropertyArray, page 689

IsChanged

Description

This property returns a Boolean value indicating whether the connected query has been changed.

Note. When an existing connected query is opened, IsChanged is initialized to False.

This property is read-only.

IsDebugMode

Description

Use this property to return a Boolean value indicating whether to perform logging while executing the connected query. In addition, in debug mode, temporary files generated during execution of the connected query are not deleted after execution is complete. The log also contains SQL statements generated by each member query as well as values for the related fields used to link the parent query to its children.

This property is read-only.

Note. While this property is read-only, it can be set through the PropertyArray property.

The default value is False.

See Also

Chapter 13, "Connected Query Classes," ExecutionLog, page 685 and Chapter 13, "Connected Query Classes," PropertyArray, page 689

IsInit

Description

This property returns a Boolean value indicating whether the connected query object has been initialized (opened) after object instance was created.

This property is read-only.

IsPublic

Description

This property returns a Boolean value indicating whether the connected query is public.

This property is read-only.

LastUpdatedBy

Description

This property returns the user ID of the user to have last updated the connected query, as a string.

This property is read-only.

LastUpdateDTTM

Description

This property returns the last update date and time for the connected query, as a datetime value.

This property is read-only.

MaxRowsPerQuery

Description

Use this property to set or return a number representing the maximum number of rows returned by each query in preview mode. If set to 0, no maximum is set.

This property is read-write.

Name

Description

This property returns the name of the connected query as a string.

This property is read-only.

ObjectRowset

Description

Use this property to set or return a Rowset object. The parent record for this rowset is PSCONQRSMAPVW; its child record is PSCONQRSFLDRLVW. Use of the ObjectRowset property and its associated rowset prevents the report developer from directly manipulating the managed object itself.

Getting the ObjectRowset property returns a Rowset object with the following structure, populated with Connected Query Manager data:

```
ObjectRowset = CreateRowset(Record.PSCONQRSMAPVW, CreateRowset⇒
(Record.PSCONQRSFLDRLVW));
```

Setting the ObjectRowset property returns the rowset data back to the Connected Query Manager.

This property is read-write.

See Also

Chapter 13, "Connected Query Classes," Iterating Through a Connected Query Opened for Editing, page 717

OprID

Description

This property returns a string representing the user ID to which the connected query belongs to for private queries. For public queries, an empty string is returned.

This property is read-only.

OutProcessFileName

Description

This property returns a string representing the output file name as an absolute path.

Note. This property is valid only for a connected query scheduled for running to a file—that is, when the run mode is equal to RunMode_Sched_File.

This property is read-only.

PropertyArray

Description

Use this property to set or return an array of CONQRSPROPERTY objects (properties) for the connected query. If no properties exist, this array returns the default set of properties.

This property is read-write.

The following read-only properties of the CONQRSMGR class are dynamically synchronized whenever the PropertyArray property is set or returned:

- ExecutionLog
- IgnoreRelFldOutput
- IsDebugMode
- ShowFormattedXML

Example

The first example retrieves a set of properties from the database and populates the page grid using the Page Activate event:


```
import PT_CONQRS:CONQRSMGR;
import PT_CONQRS:CONQRSPROPERTY;

Component PT_CONQRS:CONQRSMGR &cConQrsInst;

Local array of PT_CONQRS:CONQRSPROPERTY &propArr;
Local Rowset &rsProp;
Local number &i, &j;

&propArr = &cConQrsInst.PropertyArray;
If &propArr.Len = 0 Then
   Return;
End-If;
Local Record &recProp, &recWrk;
&rsProp = GetLevel0()(1).GetRowset(Scroll.PSCONQRSPROP);
&rsProp.Flush();
For &i = 1 To &propArr.Len
   If &i > 1 Then
      &rsProp.InsertRow(&i - 1);
   End-If;
   &recProp = &rsProp(&i).PSCONQRSPROP;
   &recProp.PROPVALUEOPTION.ClearDropDownList();
   For &j = 1 To &propArr [&i].PROPVALUEARRAY.Len
      &recProp.PROPVALUEOPTION.AddDropDownItem(&propArr [&i].PROPVALUEARRAY [&j],⇒
 &propArr [&i].PROPVALUEARRAYDESC [&j]);
   End-For;
   &recProp.PROPNAME.Value = &propArr [&i].PROPNAME;
   &recProp.PROPVALUEOPTION.Value = &propArr [&i].PROPVALUE;
   &recProp.OPRID.Value = &cConQrsInst.OprID;
   &recProp.CONQRSNAME.Value = &cConQrsInst.Name;

   &recWrk = &rsProp(&i).PSCONQRS_WRK;
   &recWrk.PROPVALUEOPTION.Value = &propArr [&i].PROPDEFAULT;
End-For;
```

The second example populates a property array from a page grid. This program is executed when a user hits the OK button on a secondary page, PSCONQRSPROPS:


```
import PT_CONQRS:CONQRSMGR;
import PT_CONQRS:CONQRSPROPERTY;

Component PT_CONQRS:CONQRSMGR &cConQrsInst;

If %Page = Page.PSCONQRSPROPS Then
   Local array of PT_CONQRS:CONQRSPROPERTY &propArr;
   Local Rowset &rsProp;
   Local number &i;
   Local Record &recProp, &recWrk;

   /* Get the property array from the database.            */
   &propArr = &cConQrsInst.PropertyArray;
   If &propArr.Len = 0 Then
      Return;
   End-If;

   &rsProp = GetLevel0()(1).GetRowset(Scroll.PSCONQRSPROP);
   If &propArr.Len <> &rsProp.ActiveRowCount Then
      Return;
   End-If;

   For &i = 1 To &rsProp.ActiveRowCount
      &recProp = &rsProp(&i).PSCONQRSPROP;
      &recWrk = &rsProp(&i).PSCONQRS_WRK;
      If None(&recProp.PROPNAME.Value) Then
         &recProp.PROPNAME.Value = &recWrk.PROPVALUEOPTION.Value;
      End-If;
      &propArr [&i].PROPNAME = &recProp.PROPNAME.Value;
      &propArr [&i].PROPVALUE = &recProp.PROPVALUEOPTION.Value;
      &propArr [&i].PROPDEFAULT = &recWrk.PROPVALUEOPTION.Value;
   End-For;

   /* Return the modified property array to the database.  */
   &cConQrsInst.PropertyArray = &propArr;
End-If;
```

See Also

Chapter 13, "Connected Query Classes," ExecutionLog, page 685; Chapter 13, "Connected Query Classes," IgnoreRelFldOutput, page 685; Chapter 13, "Connected Query Classes," IsDebugMode, page 686 and Chapter 13, "Connected Query Classes," ShowFormattedXML, page 696

Chapter 13, "Connected Query Classes," CONQRSPROPERTY Class, page 698

QueriesPromptsArray

Description

This property returns an array of QUERYITEMPROMPT objects. This array is used by the connected query object as a parameter to the Run method in preview mode.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," QUERYITEMPROMPT Class, page 705

QueryArray

Description

This property returns an array of string representing the query names that constitute the connected query object.

This property is read-only.

RegisteredBy

Description

This property returns a string representing the user ID of the user who created the connected query object.

This property is read-only.

RegisteredDTTM

Description

This property returns the creation date and time for the connected query, as a datetime value.

This property is read-only.

RunCntlId

Description

This property returns a string representing the run control ID for scheduling the connected query.

This property is read-only.

RunControlParArray

Description

This property returns an array of PSCONQRSRUNPRM records to be used for displaying and updating the set of scheduling parameters for the queries.

This property is read-only.

RunMode

Description

This property returns a number representing the run mode for a connected query instance. The values are:

Numeric Value	***Constant Value***	***Description***
1	RunMode_Window	Run the connected query in background mode with output to a separate browser window. The connected query runs via Process Scheduler and uses a REN server for delivering the output XML to the user. The XML result file is delivered to both the REN server window and the Report Manager.
2	RunMode_Sched_Web	Run the connected query at a scheduled time with output to web. The connected query is scheduled to output the XML result file to the Report Manager. It can be accessed by the user via a Report Manager link.
3	RunMode_Sched_File	Run the connected query at a scheduled time with output to a file name and location specified by the user.

Numeric Value	***Constant Value***	***Description***
4	RunMode_Prev	Run the connected query immediately in preview mode with output to a separate browser window. The connected query is executed immediately on the application server with output to a separate browser window. The output XML is not formatted internally; the only formatting that occurs is performed by the browser. Because results only exist for the browser session and are not preserved, this mode should be used during connected query creation, debugging, and tuning only. **Important!** When using the preview mode, set a limit on the number of rows returned from each query. See Chapter 13, "Connected Query Classes," MaxRowsPerQuery, page 687 and Chapter 13, "Connected Query Classes," RunMode_Prev_DefRowNumber, page 701.
5	RunMode_Sample	Run the connected query immediately with output to a file that is suitable as an XML Publisher data source.

Numeric Value	Constant Value	Description
6	RunMode_XMLFormattedString	Run the connected query immediately with output to a formatted XML string. This formatted XML string is larger than the unformatted XML of the RunMode_Prev mode. The connected query is executed immediately on the application server with output to a formatted XML string. This string is formatted according to the XML DOM object and is suitable for displaying in a long edit box. This mode is used by the Connected Query Quick Start wizard in which XML results are previewed on the application page in a long edit box. **Important!** When using the formatted XML string preview mode, set a limit on the number of rows returned from each query. See Chapter 13, "Connected Query Classes," MaxRowsPerQuery, page 687 and Chapter 13, "Connected Query Classes," RunMode_Prev_DefRowNumber, page 701.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," Run, page 677; Chapter 13, "Connected Query Classes," RunToWindow, page 678 and Chapter 13, "Connected Query Classes," RunToXMLFormattedString, page 679

Chapter 13, "Connected Query Classes," RunMode_Prev, page 701; Chapter 13, "Connected Query Classes," RunMode_Sample, page 702; Chapter 13, "Connected Query Classes," RunMode_Sched_File, page 702; Chapter 13, "Connected Query Classes," RunMode_Sched_Web, page 702; Chapter 13, "Connected Query Classes," RunMode_Window, page 703 and Chapter 13, "Connected Query Classes," RunMode_XMLFormattedString, page 703

SchedInfo

Description

Use this property to set or return a SCHED_INFO object to be populated with scheduling details. Use the property as a parameter to the Run method for scheduling connected query execution when the run mode is RunMode_Window, RunMode_Sched_Web, or RunMode_Sched_File.

This property is read-write.

See Also

Chapter 13, "Connected Query Classes," SCHED_INFO Class, page 706

SchedRequestType

Description

Use this property to set or return a string indicating which application scheduled the connected query process. The values are:

Value	*Description*
CQR	Connected Query Manager
XMLP	XML Publisher

Separate run control parameter records (PSCONQRSRUNPRM or PSCONQRSRUNPRMX) are used for different values of this property.

This property is read-write.

SecProfile

Description

This property returns a SEC_PROFILE object, which shows the user's query access level as defined for the query.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," SEC_PROFILE Class, page 708

ShowFormattedXML

Description

Use this property to return a Boolean value indicating whether to retain indentation when generating XML output.

This property is read-only.

Note. While this property is read-only, it can be set through the PropertyArray property.

The default value is False.

See Also

Chapter 13, "Connected Query Classes," PropertyArray, page 689

Status

Description

Use this property to set or return a string representing the status of the connected query. The values are:

Value	*Description*
A	Active
I	Inactive
P	In progress.

This property is read-write.

XMLDataFileName

Description

This property returns the short name for the XML data file as a string.

This property is read-only.

XMLDataFullName

Description

This property returns the long name for the XML data file as a string.

This property is read-only.

CONQRSPROPERTY Class

The CONQRSPROPERTY class is a data structure that exposes connected query properties. Individual members of the CONQRSPROPERTY class should be retrieved and set through the CONQRSMGR class, PropertyArray property, which is an array of CONQRSPROPERTY objects.

See Also

Chapter 13, "Connected Query Classes," PropertyArray, page 689

CONQRSPROPERTY Class Properties

In this section, the CONQRSPROPERTY class properties are presented in alphabetical order.

PROPDEFAULT

Description

Use this property to set or return the default property value as a string.

This property is read-write.

PROPNAME

Description

Use this property to set or return the property name as a string.

This property returns a Boolean value indicating that the specified connected query already exists. The value is False.

This property is read-write.

PROPVALUE

Description

Use this property to set or return the current property value as a string.

This property is read-write.

PROPVALUEARRAY

Description

Use this property to set or return an array of string representing the valid property values.

This property is read-write.

PROPVALUEARRAYDESC

Description

Use this property to set or return an array of string representing the descriptions for the valid property values.

This property is read-write.

CONQRS_CONST Class

This section provides an overview of the CONQRS_CONST class and discusses CONQRS_CONST class properties.

The CONQRS_CONST class is a data structure that exposes read-only properties representing constant values. An instance of the CONQRS_CONST class is created through the Const property of the CONQRSMR class.

See Also

Chapter 13, "Connected Query Classes," Const, page 684

CONQRS_CONST Class Properties

In this section, the CONQRS_CONST class properties are presented in alphabetical order.

InitExisting

Description

This property returns a Boolean value indicating that the specified connected query already exists. The value is False.

Use InitExisting as an argument to the Open method.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," Open, page 676

InitNew

Description

This property returns a Boolean value indicating that the specified connected query is new. The value is True.

Use InitNew as an argument to the Open method.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," Open, page 676

MsgSet

Description

This property returns the number 241 as the message set number for connected query.

This property is read-only.

RunCntlId_Auto

Description

This property returns the string PSCONQRS_AUTORUN.

Use this property with the SetRunControlData method for the RunMode_Window mode. It indicates that the run control ID for the connected query process should be generated automatically.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," SetRunControlData, page 681 and Chapter 13, "Connected Query Classes," RunMode_Window, page 703

RunMode_Prev

Description

This property returns the number 4, which represents the preview run mode with output to a separate browser window.

The connected query is executed immediately on the application server with output to a separate browser window. The output XML is not formatted internally; the only formatting that occurs is performed by the browser. Because results only exist for the browser session and are not preserved, this mode should be used during connected query creation, debugging, and tuning only.

Note. When using the preview mode, set a limit on the number of rows returned from each query.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," Run, page 677 and Chapter 13, "Connected Query Classes," RunMode, page 693

Chapter 13, "Connected Query Classes," MaxRowsPerQuery, page 687 and Chapter 13, "Connected Query Classes," RunMode_Prev_DefRowNumber, page 701

RunMode_Prev_DefRowNumber

Description

This property returns the number 6, which represents the default number of rows returned for connected queries running in the RunMode_Prev or RunMode_XMLFormattedString modes.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," MaxRowsPerQuery, page 687 and Chapter 13, "Connected Query Classes," RunMode_Prev, page 701

Chapter 13, "Connected Query Classes," RunMode_XMLFormattedString, page 703

RunMode_Sample

Description

This property returns the number 5, which represents a run mode that produces a sample XML string that can be used by an XML Publisher data source as a sample XML string.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," RunMode, page 693

RunMode_Sched_File

Description

This property returns the number 3, which represents the scheduled run mode with output to a file name and location specified by the user.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," Run, page 677 and Chapter 13, "Connected Query Classes," RunMode, page 693

RunMode_Sched_Web

Description

This property returns the number 2, which represents the scheduled run mode with output to the web (that is, Report Manager).

The output can be accessed by the user via a Report Manager link.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," Run, page 677 and Chapter 13, "Connected Query Classes," RunMode, page 693

RunMode_Window

Description

This property returns the number 1, which represents the background run mode with output to a separate browser window.

The connected query runs via Process Scheduler and uses a REN server for delivering the output XML to the user. The XML result file is delivered to both the REN server window and the Report Manager.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," RunToWindow, page 678 and Chapter 13, "Connected Query Classes," RunMode, page 693

RunMode_XMLFormattedString

Description

This property returns the number 6, which represents the run mode that produces an formatted XML string. This formatted XML string is larger than the unformatted XML of the RunMode_Prev mode.

The connected query is executed immediately on the application server with output to a formatted XML string. This string is formatted according to the XML DOM object and is suitable for displaying in a long edit box. This mode is used by the Connected Query Quick Start wizard in which XML results are shown on the application page in a long edit box.

Note. When using the preview mode, set a limit on the number of rows returned from each query.

This property is read-only.

See Also

Chapter 13, "Connected Query Classes," RunToXMLFormattedString, page 679 and Chapter 13, "Connected Query Classes," RunMode, page 693

Chapter 13, "Connected Query Classes," MaxRowsPerQuery, page 687 and Chapter 13, "Connected Query Classes," RunMode_Prev_DefRowNumber, page 701

SchedRequest_CQR

Description

This property returns the string CQR representing that connected query execution was invoked by the Connected Query Manager. The value of this property is used as a part of the SCHED_INFO data structure.

This property is read-only.

SchedRequest_XMLP

Description

This property returns the string XMLP, representing that connected query execution was invoked by XML Publisher. The value of this property is used as a part of the SCHED_INFO data structure.

This property is read-only.

Stat_Active

Description

This property returns the string A indicating that the connected query definition is active.

This property is read-only.

Stat_InActive

Description

This property returns the string I indicating that the connected query definition is inactive. A connected query can be set to inactive if it did not pass a validation.

This property is read-only.

Stat_InProgress

Description

This property returns the string P indicating that the connected query definition is in progress. A connected query that is "in process" be run in preview mode only. It cannot be scheduled, and it is not be visible to XML Publisher.

This property is read-only.

QUERYITEMPROMPT Class

This section provides an overview of the QUERYITEMPROMPT class and discusses QUERYITEMPROMPT class properties.

The QUERYITEMPROMPT class is a data structure that exposes read-only properties representing query prompts. An array of instances of this class is exposed via the QueriesPromptsArray property of the CONQRSMR class. Therefore, only an import statement is required, and no constructor:

```
import PT_CONQRS:QUERYITEMPROMPT;
```

See Also

Chapter 13, "Connected Query Classes," QueriesPromptsArray, page 691

Chapter 6, "Application Classes," Import Declarations, page 209

QUERYITEMPROMPT Class Properties

In this section, the QUERYITEMPROMPT class properties are presented in alphabetical order.

QueryName

Description

This property returns as string representing the name of a query that participates in the connected query.

This property is read-only.

QueryPromptRecord

Description

This property returns a Record object with the runtime prompts for the query.

This property is read-only.

SCHED_INFO Class

This section provides an overview of the SCHED_INFO class and discusses SCHED_INFO class properties.

The SCHED_INFO class is a data structure that exposes the class constructor and read-write properties representing run controls and scheduling information. An instance of this class is exposed via the SchedInfo property of the CONQRSMR class. Therefore, only an import statement is required, and no constructor:

```
import PT_CONQRS:SCHED_INFO;
```

See Also

Chapter 13, "Connected Query Classes," SchedInfo, page 695

Chapter 6, "Application Classes," Import Declarations, page 209

SCHED_INFO Class Properties

In this section, the SCHED_INFO class properties are presented in alphabetical order.

AE_ID

Description

Use this property to set or return a string indicating which application scheduled execution of the connected query. The values are:

Value	*Description*
CQR (the default)	Connected Query Manager
XMLP	XML Publisher

This property is read-write.

DIRLOCATION

Description

Use this property to set or return a string representing the directory location for temporary files generated during processing of a scheduled connected query. This directory should be the same as the log/output directory specified in the psprcs.cfg file, allowing the connected query to use the distribution features of Process Scheduler.

This property is read-write.

OPRID

Description

Use this property to set or return a string representing the ID of the user who initiated the process.

This property is read-write.

OUTDESTTYPE

Description

Use this property to set or return the Process Scheduler output destination type as a number. The values are:

Value	***Description***
2	File
4	Window
6	Web

This property is read-write.

PRCSFILENAME

Description

Use this property to set or return the output file name as a string. This property is required when OUTDESTTYPE equals 2 (file). Specify an absolute path and file name that is accessible by the Process Scheduler server.

This property is read-write.

PROCESS_INSTANCE

Description

Use this property to set or return the process instance as a string.

This property is read-write.

RUN_CNTL_ID

Description

Use this property to set or return the run control ID as a string.

This property is read-write.

SEC_PROFILE Class

This section provides an overview of the SEC_PROFILE class and discusses:

- SEC_PROFILE class constructor.
- SEC_PROFILE class properties.

The SEC_PROFILE class is a data structure that exposes the class constructor and read-only properties representing the security access to the connected query for this user's session.

See Also

Chapter 13, "Connected Query Classes," SecProfile, page 696

SEC_PROFILE Class Constructor

This section presents the constructor for the SEC_PROFILE class.

SEC_PROFILE

Example

```
import PT_CONQRS:SEC_PROFILE;
```

```
Local PT_CONQRS:SEC_PROFILE &SecProfile = create PT_CONQRS:SEC_PROFILE();
```

See Also

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

SEC_PROFILE Class Properties

In this section, the SEC_PROFILE class properties are presented in alphabetical order.

CanCreatePublic

Description

This property returns a Boolean value indicating whether the user can create a public connected query.

This property is read-only.

CanModify

Description

This property returns a Boolean value indicating whether the user can modify the connected query.

This property is read-only.

CanRunQuery

Description

This property returns a Boolean value indicating whether the user can run the connected query.

This property is read-only.

UTILITY Class

This section provides an overview of the UTILITY class and discusses:

- UTILITY class constructor.
- UTILITY class methods.

The UTILITY class is a data structure that exposes the class constructor and utility methods used for validation. The UTILITY class does not require that a connected query be opened first.

UTILITY Class Constructor

This section presents the constructor for the UTILITY class.

UTILITY

Example

```
import PT_CONQRS:UTILITY;

Local PT_CONQRS:UTILITY &Utility = create PT_CONQRS:UTILITY();
```

See Also

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

UTILITY Class Methods

In this section, the UTILITY class methods are presented in alphabetical order.

CheckQryForTreePrompt

Syntax

CheckQryForTreePrompt(*QryName*, *scope*)

Description

Use this method to checks if a specific query uses tree prompts.

Important! Connected query does not support queries with tree prompts.

Parameters

Parameter	***Description***
QryName	Specifies the query name as a string.
scope	Specifies the query scope as a string. If a valid value is not specified, CheckQryForTreePrompt searches to determine the scope. The values are: • R — Private • U — Public

Returns

The empty string if the query does not use tree prompts; otherwise, an error string.

CheckQrySecurity

Syntax

`CheckQrySecurity`*`(QryName,IsPublicObject)`*

Description

Use this method to check user access security for a single query. CheckQrySecurity performs the following checks:

- Whether the query is public for public connected queries *(IsPublicObject* is True).
- Whether the query is disabled.
- Whether the query has tree prompts by invoking CheckQryForTreePrompt.
- Whether there are query prompt security violations.

Parameters

Parameter	*Description*
QryName	Specifies the query name as a string.
IsPublicObject	Specifies a Boolean value indicating whether the connected query is public.

Returns

A Boolean value: True if the security validation was successful, False otherwise.

GetQueryScopeByName

Syntax

`GetQueryScopeByName`*`(QryName)`*

Description

Use this method to check whether the specified query has a public or private scope.

Parameters

Parameter	*Description*
QryName	Specifies the query name as a string.

Returns

A number:

- 0 — Private query.
- 1 — Public query.
- 2 — Query not found.

ValidateObjectID

Syntax

```
ValidateObjectID(ID)
```

Description

Use this method to check whether the specified ID (connected query ID or run control ID) contains special characters. Valid characters for IDs include A-Z, 0–9, and _.

Parameters

Parameter	***Description***
ID	Specifies the ID to be validated as a string.

Returns

An empty string if the validation was successful; otherwise, an error string.

Connected Query Classes Examples

This section includes examples of:

- Running a connected query in scheduled mode.
- Running a connected query in preview mode or background mode.
- Iterating through a connected query opened for editing.

Running a Connected Query in Scheduled Mode

In scheduled mode, the connected query is run by Process Scheduler. In the case of RunMode_Sched_File, the output data is written to an XML file. In the case of RunMode_Sched_Web, the output data is accessible via a hyperlink in Process Monitor. In scheduled mode, the *runProcessInfo* parameter is required; the *Prompts* parameter should be set to Null.

The following example represents a PeopleCode step to be run via Application Engine:

```
import PT_CONQRS:CONQRSMGR;
import PT_CONQRS:SCHED_INFO;

Local PT_CONQRS:CONQRSMGR &cConQrsInst;
Local boolean &result;
Local string &PRCSFILENAME;
Local string &sPrcsName = "PSCONQRS";
Local string &sPrcsType = "Application Engine";
Local number &nOrigPSMessagesMode = %Session.PSMessagesMode;
%Session.PSMessagesMode = 1;

/* While working with connected queries, it is recommended to use a try- */
/* catch block                                                           */

try

   /* Create and open an object. System tries to find a connected query  */
   /* with private ownership first for current user. If not found, it    */
   /* uses a public ownership                                            */

   &ConQrsInst = create PT_CONQRS:CONQRSMGR("", PSCONQRS_AET.CONQRSNAME);

   &result = &ConQrsInst.Open(&ConQrsInst.Const.InitExisting);

   &str = &ConQrsInst.ErrString;
   If &str <> "" Then
      WriteToLog(%ApplicationLogFence_Error, &ConQrsInst.ErrString);
      %Session.PSMessagesMode = &nOrigPSMessagesMode;
      Exit (1);
   End-If;


   If &result Then

      /* Get an empty SCHEDINFO structure and populate it.               */
      &schedInfo = &ConQrsInst.SchedInfo;
      &schedInfo.DIRLOCATION = %FilePath;
      &schedInfo.OUTDESTTYPE = String(%OutDestType);
      &schedInfo.RUN_CNTL_ID = PSCONQRS_AET.RUN_CNTL_ID;
      &schedInfo.PROCESS_INSTANCE = PSCONQRS_AET.PROCESS_INSTANCE;
      &schedInfo.OPRID = PSCONQRS_AET.OPRID;
      If %OutDestType = 2 Then /* FILE */
         Local string &sqlFile = "SELECT OUTDEST FROM PS_PRCSRUNCNTLDTL ⇒
WHERE OPRID=:1 and RUNCNTLID =:2 and PRCSTYPE =:3 and PRCSNAME =:4";
         SQLExec(&sqlFile, PSCONQRS_AET.OPRID, PSCONQRS_AET.RUN_CNTL_ID, ⇒
&sPrcsType, &sPrcsName, &PRCSFILENAME);
      End-If;
      &schedInfo.PRCSFILENAME = &PRCSFILENAME;

      /* Indicate the source of the scheduled request as CQR.            */
      &schedInfo.AE_ID = &ConQrsInst.Const.SchedRequest_CQR;

      /* Schedule the query to run. Note that for scheduled connected    */
      /* queries, the Prompts parameter is Null.                         */
      &result = &ConQrsInst.Run( Null, &schedInfo);
   End-If;
   If &result Then
      %Session.PSMessagesMode = &nOrigPSMessagesMode;
      Exit (0);
   Else

      /* Check for errors  */
      &str = &ConQrsInst.ErrString;
```



```
      If &str <> "" Then
         WriteToLog(%ApplicationLogFence_Error, &ConQrsInst.ErrString);
      End-If;
      /* check session message for errors */
      If %Session.PSmessages.Count > 0 Then
         &PSMessages = %Session.PSmessages;
         For &i = 1 To &PSMessages.Count
            If (&PSMessages.Item(&i).MessageType <= 1) Then
               &MsgSetNbr = &PSMessages.Item(&i).MessageSetNumber;
               &MsgNbr = &PSMessages.Item(&i).MessageNumber;
               WriteToLog(%ApplicationLogFence_Error, MsgGet(&MsgSetNbr, ⇒
&MsgNbr, "Message Not Found : " | &MsgSetNbr | "," | &MsgNbr));
            End-If;
         End-For;
      End-If;
   End-If;

   %Session.PSMessagesMode = &nOrigPSMessagesMode;

catch Exception &Err
   WriteToLog(%ApplicationLogFence_Error, &Err.ToString());
end-try;
```

Running a Connected Query in Preview Mode or Background Mode

In preview mode, the connected query is run immediately on the application sever. The output data is displayed in a separate browser window. In preview mode, the *Prompts* parameter is required; the *runProcessInfo* parameter *must* be set to Null. The *Prompts* parameter should be retrieved using the QueriesPromptsArray property. If this array length is greater than 0, this array must be populated.

Similar to preview mode, running a connected query in background mode results in the output data being displayed in a separate browser window. However, in background mode, the connected query is run through Process Scheduler with a REN server. Running a connected query in this manner prevents blocking on the application server. While the report file is generated, the user can continue working with the PeopleSoft application.

In the following example, assume that an application calls the RunConnectedQuery function with all required parameters. The function returns an error string if any error occurs.


```
import PT_CONQRS:CONQRSMGR;
import PT_CONQRS:QUERYITEMPROMPT;

Function ShowException(&errString As Exception);

   /* Perform exception processing  */

End-Function;

Function RunConnectedQuery(&sOprId As string, &sCONQRSNAME As string, &RunMode ⇒
As number) Returns string

   Local boolean &result;
   Local array of PT_CONQRS:QUERYITEMPROMPT &aPrompts;
   Local string &str;
   Local number &processID;
   Local boolean &IsChangeDB, &IsNewCnt;
   Local PT_CONQRS:CONQRSMGR &cConQrsInst;

   /* While working with connected queries, it is recommended to use a try- */
   /* catch block                                                           */

   try

      /* Create and open an object. System tries to find a connected query  */
      /* with private ownership first for current user. If not found, it    */
      /* uses a public ownership                                            */

      &cConQrsInst = create PT_CONQRS:CONQRSMGR("", &sCONQRSNAME);
      &result = &cConQrsInst.Open(&cConQrsInst.Const.InitExisting);

      /* Check for errors  */
      &str = &cConQrsInst.ErrString;
      If &str <> "" Then
         Error &cConQrsInst.ErrString;
      End-If;

      /* Validate the previously opened connected query object              */
      If Not &cConQrsInst.Validate() Then
         &str = &cConQrsInst.ErrString;
         If &str <> "" Then
            Error &cConQrsInst.ErrString;
         End-If;
      End-If;

      /* Preview Mode- AppServer execution */
      /* Populate prompts. NOTE: It is required for 'interactive' execution */
      /* using an application server. In the case of scheduled execution,   */
      /* prompts are read from the database                                 */
      If &RunMode = &cConQrsInst.Const.RunMode_Prev Then
         &aPrompts = FillQueriesPrompts();
         If None(&aPrompts) Then
            /* User hit Cancel on a Prompt dialog */
            &cConQrsInst.ResetQueriesPrompt();
            Return &cErrIndicator;
         End-If;

         /* Restrict number of rows being displayed for each member query.  */
         /* If user does not set this value, the default number is used     */
         If All(PSCONQRS_WRK.QRY_MAX_FETCH) Then
            &cConQrsInst.MaxRowsPerQuery = &numRows;
         Else
            &cConQrsInst.MaxRowsPerQuery = &cConQrsInst.Const.⇒
```



```
RunMode_Prev_DefRowNumber;
         End-If;

         /* Run the connected query. NOTE: The runProcessInfo parameter is Null */
         &result = &cConQrsInst.Run(&aPrompts, Null);

      End-If; /* end of Mode selection */


      /* 'Run to Window' mode */
      If &RunMode = &cConQrsInst.Const.RunMode_Window Then
         &IsChangeDB = True;
         &IsNewCnt = False;

         /* You are required to set run control parameters for immediate    */
         /* execution using Process Scheduler                                 */
         &result = &cConQrsInst.SetRunControlData(&cConQrsInst.Const.⇒
RunCntlId_Auto, &IsNewCnt, &IsChangeDB);
         If &result Then
            &processID = &cConQrsInst.RunToWindow();
            If All(&processID) Then
               &result = True;;
            End-If;
         End-If;
      End-If; /* end of Mode selection */

      /* Check for errors  */
      &str = &cConQrsInst.ErrString;
      If &str <> "" Then
         Error &cConQrsInst.ErrString;
      End-If;
      Return &str;

   catch Exception &Err
      ShowException(&Err);
   end-try;

End-Function;
```

Iterating Through a Connected Query Opened for Editing

This example demonstrates usage of the connected query ObjectRowset property. The program opens an instance from an existing connected query definition. The program retrieves the rowset from the ObjectRowset property, and changes some of its properties. It saves the rowset back to ObjectRowset property, and then saves the modified connected query definition back to the database.


```
import PT_CONQRS:CONQRSMGR;

Local PT_CONQRS:CONQRSMGR &cConQrsInst;
Local string &sOprId, &sCONQRSNAME, &str, &sQryParentName, &sQryChildName, ⇒
&sFldNamePar, &sFldNameChild;
Local Rowset &rsObjectRowset, &rsObjectFields;
Local Record &rMapObjRec, &rFldObjRec;
Local number &i, &j, &seqNum, &fldNum;

Function ShowException(&errString As Exception);

End-Function;

/* While working with connected queries, it is recommended to use a try- */
/* catch block                                                           */

try

   /* Create and open an object. System tries to find a connected query  */
   /* with private ownership first for current user. If not found, it    */
   /* uses a public ownership                                            */

   &sCONQRSNAME = "MyConQuery";
   &cConQrsInst = create PT_CONQRS:CONQRSMGR("", &sCONQRSNAME);

   &res = &cConQrsInst.Open(&cConQrsInst.Const.InitExisting);
   &str = &cConQrsInst.ErrString;

   If &str <> "" Then
      Error &cConQrsInst.ErrString;
   End-If;

   /* Validate the previously opened connected query object              */
   If Not &cConQrsInst.Validate() Then
      &str = &cConQrsInst.ErrString;
      If &str <> "" Then
         Error &cConQrsInst.ErrString;
      End-If;
   End-If;

   /* Rowset retrieved from a connected query has the following          */
   /* structure: ObjectRowset = CreateRowset(Record.PSCONQRSMAPVW,       */
   /* CreateRowset(Record.PSCONQRSFLDRLVW));                             */

   &rsObjectRowset = &cConQrsInst.ObjectRowset;

   /* The rowset contains the internal connected query structure. The    */
   /* user can iterate the rowset, retrieve and modify member query      */
   /* names, list the related fields, list object properties, etc.       */

   For &i = 1 To &rsObjectRowset.ActiveRowCount
      &rMapObjRec = &rsObjectRowset(&i).PSCONQRSMAPVW;
      &sQryParentName = &rMapRec.GetField(Field.QRYNAMEPARENT).Value;
      &sQryChildName = &rMapRec.GetField(Field.QRYNAMECHILD).Value;
      &seqNum = &rsObjectRowset(&i).PSCONQRSMAPVW.SEQNUM.Value;

      &rsObjectFields = &rsObjectRowset(&i).GetRowset(Scroll.PSCONQRSFLDRLVW);
      For &j = 1 To &rsObjectFields.ActiveRowCount
         &rFldObjRec = &rsObjectFields(&j).PSCONQRSFLDRLVW;
         &fldNum = &rFldObjRec.SELCOUNT.Value;
         &sFldNamePar = &rFldObjRec.QRYFLDNAMEPAR.Value;
         &sFldNameChild = &rFldObjRec.QRYFLDNAMECHILD.Value;;
      End-For;
   End-For;
```



```
   /* The preceding for loop shows how to navigate a connected query's   */
   /* internal structure. It allows a report developer to change a       */
   /* connected query definition by modifying existing query members or  */
   /* by adding new ones -- queries or related fields.                   */


   /* Need to set the connected query's rowset to the retrieved rowset   */
   /* after any changes are made to the retrieved rowset.                */

   &cConQrsInst.ObjectRowset = &rsObjectRowset;

   &Errstr = &cConQrsInst.ErrString;
   If &Errstr <> "" Then
      Error &cConQrsInst.ErrString;
   End-If;

   /* Set the connected query properties. Set the object's status to     */
   /* active, so it will be available for reporting use                  */

   &cConQrsInst.Description = "My Connected Query";
   &cConQrsInst.Details = "Includes QryA, QryA1, and QryC";
   &cConQrsInst.Status = "A";

   /* Save the connected query instance                                  */
   If Not &cConQrsInst.Save() Then
      If &cConQrsInst.ErrString <> "" Then
         Error &cConQrsInst.ErrString;
      End-If;
   End-If;

catch Exception &Err
   ShowException(&Err);
end-try;
```


Chapter 14

Crypt Class

This chapter provides an overview of the crypt class and discusses:

- Create a crypt object
- Error handling
- Data type of a crypt object
- Scope of a crypt object
- Crypt class reference

Understanding the Crypt Class

The crypt class is used with pluggable cryptography. After you create an encryption profile, use PeopleCode to invoke the encryption profile for encrypting, decrypting, or signing a field, depending on the profile

See Also

PeopleTools 8.51 PeopleBook: Security Administration, "Securing Data with PeopleSoft Encryption Technology"

Creating a Crypt Object

The crypt class does not have a separate function for instantiating an object (such as CreateCrypt.) Instead, you instantiate a crypt object using the CreateObject function, using the keyword **Crypt.**

```
&cry = CreateObject("Crypt");
```

See *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," CreateObject.

Declaring Crypt Objects

Crypt objects are declared by using the Crypt type name.


```
Local Crypt &MyCrypt;
```

Note. Crypt objects cannot be serialized, and so can only be declared as Local.

Scope of a Crypt Object

A crypt object can only be instantiated from PeopleCode. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

Crypt Class Methods

In this section, we discuss the crypt class methods. The methods are discussed in alphabetical order.

FirstStep

Syntax

FirstStep()

Description

Use the FirstStep method to access the first step in the encryption chain.

You must use either the FirstStep or GoToStep methods before you use the NextStep or SetParameter methods.

Parameters

None.

Returns

None.

See Also

Chapter 14, "Crypt Class," GoToStep, page 723; Chapter 14, "Crypt Class," NextStep, page 724 and Chapter 14, "Crypt Class," SetParameter, page 725

GoToStep

Syntax

`GoToStep`*`(StepNum)`*

Description

Use the GoToStep method to access a specific step in the encryption chain.

You must use either the GoToStep or FirstStep methods before you use the NextStep or SetParameter method.

Parameters

Parameter	***Description***
StepNum	Specify the step number that you want to access, as a number.

Returns

None.

See Also

Chapter 14, "Crypt Class," FirstStep, page 722; Chapter 14, "Crypt Class," NextStep, page 724 and Chapter 14, "Crypt Class," SetParameter, page 725

LoadLibrary

Syntax

`LoadLibrary`*`(LibraryFile,LibraryID)`*

Description

Use the LoadLibrary method to specify the encryption library to be used. This method is generally used when either your underlying library changes (such as, a new version, added algorithms, and so on) or you have your written own library and you need to load the metadata into the PeopleSoft system.

Parameters

Parameter	*Description*
LibraryFile	Specify the name of the file containing the encryption library as a string. You do not have to specify a full path name. The delivered OpenSSL library is pspetssl.dll. The delivered PGP library is pspetpgp.dll.
LibraryID	Specify the name of the library, as a string.

Returns

None.

NextStep

Syntax

NextStep()

Description

Use the NextStep method to access the next step in the encryption chain.

You must use the FirstStep or GoToStep method before using NextStep.

Your program terminates if you call NextStep when you are already at the last step in the encryption chain.

Parameters

None.

Returns

None.

See Also

Chapter 14, "Crypt Class," FirstStep, page 722 and Chapter 14, "Crypt Class," GoToStep, page 723

Open

Syntax

Open*(ProfileName)*

Description

Use the Open method to open the encryption profile identified by *ProfileName.* You must open an encryption profile before you can add data to the encryption profile. Your program terminates if you specify an encryption profile that doesn't exist.

Parameters

Parameter	***Description***
ProfileName	Specify the name of the encryption profile you want to access, as a string. You can store the name of the encryption profile in a field, and specify a *fieldname.recordname.*

Returns

None.

Example

```
Local Crypt &cry;

&cry = CreateObject("Crypt");
&bar = QE_CRYPT_WRK.CRYPT_PRFL_ID;
&cry.Open(&bar);
&cry.UpdateData(QE_CRYPT_WRK.DESCRLONG);
QE_CRYPT_WRK.LARGECHAR = &cry.Result;
```

SetParameter

Syntax

SetParameter*(Name,Value)*

Description

Use the SetParameter method to set the parameter specified by *Name* to a value specified by *Value.*

You must have already used the FirstStep, NextStep, or GoToStep methods to specify a step before using this method.

Parameters

Parameter	Description
Name	Specify the name of the parameter that you want to change, as a string.
Value	Specify the value for the parameter that you want to change.

Returns

None.

See Also

Chapter 14, "Crypt Class," FirstStep, page 722; Chapter 14, "Crypt Class," GoToStep, page 723 and Chapter 14, "Crypt Class," NextStep, page 724

UpdateData

Syntax

`UpdateData`*`(Data)`*

Description

Use the UpdateData method to add data to the encryption chain. This method can be called multiple times after opening a profile to add data.

Parameters

Parameter	Description
Data	Specify the data you want to add to the encryption chain, as a string.

Returns

None.

Crypt Class Properties

This section describes the crypt class properties. The properties are described in alphabetical order.

Result

Description

After updating the encryption chain, the Result property contains the end result of the encryption chain. Once the result has been retrieved, it is no longer possible to update the object anymore.

This property is read-only.

Verified

Description

For algorithms that check a signature, the system sets the Verified property to true if the signature is valid, and false if the signature is invalid. For algorithms that do not check a signature, it always returns false.

This property is read-only.

Chapter 15

Document Classes

This chapter provides an overview of the document classes and discusses:

- Data type and scope of the document classes.
- Document classes built-in functions.
- Document class.
- DocumentKey class.
- Primitive class.
- Compound class.
- Collection class.

Understanding the Document Classes

A PeopleSoft document is a managed object that has two representations. The logical document depicts the document's structure. The physical document is the rendering of the document in a specific format—either in XML or as a relational record (or records).

PeopleCode provides document classes that contain built-in functions and methods that enable you to populate data into and retrieve data from logical documents. However, you should already have knowledge of the logical document structure in order to effectively write PeopleCode to work with a specific document.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Documents Technology, "Understanding PeopleSoft Documents Technology"

Data Type and Scope of the Document Classes

Every document class is its own data type—that is, documents are declared as the Document data type, primitive objects are declared as the Primitive type, and so on. For example:

```
Local Document &Doc;
```

The following are the data types of the document classes:

- Document
- DocumentKey
- Primitive
- Compound
- Collection

Document objects can only be instantiated from PeopleCode. These objects can be used anywhere you have PeopleCode—that is, in Component Interface PeopleCode, record field PeopleCode, Application Engine PeopleCode, and so on.

Document Classes Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateDocument

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateDocumentKey

Document Class

This section provides an overview of the Document class and discusses:

- Document class methods.
- Document class properties.

The Document class provides the ability to manipulate a PeopleSoft document as a Document object.

Document Class Methods

In this section, the Document class methods are presented in alphabetical order.

GenXmlString

Syntax

`GenXmlString``([`*`validate`*`])`

Description

Use this method to generate an XML string representing the logical document populated with data.

Note. The Generate button on the Document Tester page uses this method to generate a test XML document.

Parameters

Parameter	***Description***
validate	Specifies a Boolean value indicating whether to validate the XML document before it is generated.

Returns

An XML string.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Documents Technology, "Testing Documents," Generating and Viewing Test Documents

GetDocumentKey

Syntax

`GetDocumentKey``()`

Description

Use this method to generate the document keys for the document as a DocumentKey object.

Parameters

None.

Returns

A DocumentKey object.

Example

```
Local Document &Doc;

&Doc = CreateDocument("Purchasing", "PurchaseOrder", "v1");
&DocKey = &Doc.GetDocumentKey();
```

See Also

Chapter 15, "Document Classes," DocumentKey Class, page 739

GetElement

Syntax

`GetElement``(`*`ElementName`*`)`

Description

Use this method to retrieve a document element by name. The object returned will be a Collection, Compound, or Primitive object.

Note. Use the object's ElementType property to determine which object type is returned.

Parameters

Parameter	***Description***
ElementName	Specifies the name of the document element as a string.

Returns

A Collection object, a Compound object, or a Primitive object.

Example

```
&Doc = CreateDocument(&DocKey);
&DocElem = &Doc.GetElement("BillTo");

Evaluate &DocElem.ElementType

When = %Document_Compound
   ...

When = %Document_Collection
   ...

When = %Document_Primitive
   ...

When-Other
   Error ...

End-Evaluate;
```

See Also

Chapter 15, "Document Classes," ElementType, page 759

Chapter 15, "Document Classes," ElementType, page 751

Chapter 15, "Document Classes," ElementType, page 743

GetRowset

Syntax

GetRowset()

Description

Use this method to generate a standalone rowset for a relational-formatted document. The structure of the rowset matches the structure of the record mapped to the document. If the document is populated with data when GetRowset is called, then the rowset is populated with that data; otherwise, the rowset is empty.

Parameters

None.

Returns

A Rowset object.

Example

```
Local Document &Doc;
Local Rowset &Doc_RS;

&Doc = CreateDocument(&DocKey);
&Doc_RS = &Doc.GetRowset();
```

See Also

Chapter 15, "Document Classes," UpdateFromRowset, page 737

Chapter 36, "Rowset Class," page 2127

GetSchema

Syntax

GetSchema()

Description

Use this method to generate the XML schema definition for the document.

Parameters

None.

Returns

A string containing the XML schema definition.

Example

```
Local Document &Doc;
Local DocumentKey &DocKey;

&DocKey = CreateDocumentKey("Purchasing", "PurchaseOrder", "v1");
&Doc = CreateDocument(&DocKey);
&str = &Doc.GetSchema();
```


ParseXmlFromFile

Syntax

`ParseXmlFromFile``(`*`file_name`*` [, `*`validate`*`])`

Description

Use this method to generate a document from the XML schema definition provided in a file.

Parameters

Parameter	***Description***
file_name	Specifies the name of the file as a string.
validate	Specifies a Boolean value indicating whether to validate the XML document as it is generated.

Returns

A Boolean value: True if a document (or a valid document) was generated, False otherwise.

See Also

Chapter 15, "Document Classes," ParseXmlFromURL, page 735 and Chapter 15, "Document Classes," ParseXmlString, page 736

ParseXmlFromURL

Syntax

`ParseXmlFromURL``(`*`URL`*` [, `*`validate`*`])`

Description

Use this method to generate a document from the XML schema definition provided at the specified URL.

Parameters

Parameter	*Description*
URL	Specifies the URL as a string.
validate	Specifies a Boolean value indicating whether to validate the XML document as it is generated.

Returns

A Boolean value: True if a document (or a valid document) was generated, False otherwise.

See Also

Chapter 15, "Document Classes," ParseXmlFromFile, page 735 and Chapter 15, "Document Classes," ParseXmlString, page 736

ParseXmlString

Syntax

ParseXmlString(*XML_string* [, *validate*])

Description

Use this method to generate a document from the XML schema definition provided as an XML string.

Parameters

Parameter	*Description*
XML_string	Specifies the XML as a string.
validate	Specifies a Boolean value indicating whether to validate the XML document as it is generated.

Returns

A Boolean value: True if a document (or a valid document) was generated, False otherwise.

See Also

Chapter 15, "Document Classes," ParseXmlFromFile, page 735 and Chapter 15, "Document Classes," ParseXmlFromURL, page 735

UpdateFromRowset

Syntax

UpdateFromRowset(*&Doc_RS*)

Description

Use this method to update the document data with data from a stand-alone rowset.

Important! Any existing data in the document is erased and replaced by the data from the rowset.

Parameters

Parameter	*Description*
&Doc_RS	Specifies the stand-alone rowset as a Rowset object.

Returns

A Boolean value: True if the update was successful, False otherwise.

See Also

Chapter 15, "Document Classes," GetRowset, page 733

Chapter 36, "Rowset Class," page 2127

ValidateData

Syntax

ValidateData()

Description

Use this method to validate the data in a document.

Parameters

None.

Returns

A Boolean value: True if the document data is valid, False otherwise.

Document Class Properties

In this section, the Document class properties are presented in alphabetical order.

DocumentElement

Description

Use this property to return the root element for this document as a Compound object.

This property is read-only.

Example

```
Local Document &Doc;
Local DocumentKey &DocKey;
Local Compound &Root


/* Instatiate the Document object */
&DocKey = CreateDocumentKey("Purchasing", "PurchaseOrder", "v1");
&Doc = CreateDocument(&DocKey);

&Root = &Doc.DocumentElement;
```

See Also

Chapter 15, "Document Classes," Compound Class, page 748

DocumentKey Class

This section provides an overview of the DocumentKey class and discusses:

- DocumentKey class methods.
- DocumentKey class properties.

The DocumentKey class provides the ability to pass a document's keys as a single object.

DocumentKey Class Methods

In this section, the DocumentKey class methods are presented in alphabetical order.

SetDocumentKey

Syntax

`SetDocumentKey``(`*`Package,DocumentName,Version`*`)`

Description

Use this method to change the document keys for a DocumentKey object.

Parameters

Parameter	***Description***
Package	Specifies a document package as a string.
DocumentName	Specifies the name of the document as a string. **Note.** The document name also becomes the root element name for the document.
Version	Specifies the document version as a string.

Returns

A Boolean value: True if the document keys were set successfully, False otherwise.

Example

```
Local Document &Doc;
Local DocumentKey &DocKey;

/* Instatiate the Document object */
&DocKey = CreateDocumentKey("Purchasing", "PurchaseOrder", "v1");

   ...

&ret = &DocKey.SetDocumentKey("Purchasing", "PurchaseOrder", "v1.1");
&Doc = CreateDocument(&DocKey);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateDocumentKey

DocumentKey Class Properties

In this section, the DocumentKey class properties are presented in alphabetical order.

DocumentName

Description

Use this property to return the document name from the document keys as a string.

This property is read-only.

PackageName

Description

Use this property to return the package name from the document keys as a string.

This property is read-only.

Version

Description

Use this property to return the version from the document keys as a string.

This property is read-only.

Primitive Class

This section provides an overview of the Primitive class and discusses:

- Primitive class methods.
- Primitive class properties.

The Primitive class provides the ability to work with a primitive document element.

Primitive Class Methods

In this section, the Primitive class methods are presented in alphabetical order.

GetEnumName

Syntax

`GetEnumName``(`*`index`*`)`

Description

Use this method to return the name of the enumerated value.

Parameters

Parameter	***Description***
index	Specifies the index of the enumerated value.

Returns

A string representing the enumerated value.

See Also

Chapter 15, "Document Classes," EnumCount, page 743

GetParent

Syntax

GetParent()

Description

Use this method to return the parent object of the primitive element.

Parameters

None.

Returns

A Collection object or a Compound object.

See Also

Chapter 15, "Document Classes," Collection Class, page 753

Chapter 15, "Document Classes," Compound Class, page 748

GetPath

Syntax

GetPath()

Description

Use this method to return the absolute path to the primitive within the document's structure.

For example, for the Name primitive of the ShipTo compound of the PuchaseOrder document, GetPath would return the following absolute path:

```
PurchaseOrder/ShipTo/Name
```

Parameters

None.

Returns

A string representing the absolute path to the primitive.

Primitive Class Properties

In this section, the Primitive class properties are presented in alphabetical order.

ElementType

Description

Use this property to return the type of this document element as an integer constant: %Document_Primitive,

This property is read-only.

EnumCount

Description

Use this property to return the number of enumerated values for this primitive element as an integer.

This property is read-only.

See Also

Chapter 15, "Document Classes," GetEnumName, page 741

IsChanged

Description

Use this property to return a Boolean value indicating whether the value of this primitive element has been changed.

This property is read-only.

See Also

Chapter 15, "Document Classes," OrigValue, page 745 and Chapter 15, "Document Classes," Value, page 747

IsInitialized

Description

Use this property to return a Boolean value indicating whether this primitive element has an initial value.

This property is read-only.

See Also

Chapter 15, "Document Classes," OrigValue, page 745 and Chapter 15, "Document Classes," Value, page 747

IsRequired

Description

Use this property return a Boolean value indicating whether this primitive element is required.

This property is read-only.

MaxDefinedDecimalLength

Description

Use this property to return the number of allowed spaces to the right of the decimal point as an integer.

Note. This property is available and valid for a decimal primitive type only.

This property is read-only.

See Also

Chapter 15, "Document Classes," MaxDefinedLength, page 744 and Chapter 15, "Document Classes," PrimitiveType, page 746

MaxDefinedLength

Description

Use this property to return the length of the primitive element as an integer.

This property is read-only.

See Also

Chapter 15, "Document Classes," MaxDefinedDecimalLength, page 744

Name

Description

Use this property to return the name of this element as a string.

This property is read-only.

OrigValue

Description

Use this property to set or return the original value for this element as a string.

This property is read-write.

See Also

Chapter 15, "Document Classes," IsChanged, page 743; Chapter 15, "Document Classes," IsInitialized, page 744 and Chapter 15, "Document Classes," Value, page 747

PrimitiveSubType

Description

Use this property to return the subtype for this primitive element as an integer. Only certain primitive types have defined subtypes.

For the %Document_Integer primitive type, the values are:

Numeric Value	***Constant Value***	***Description***
—	*none*	No subtype (default).
1	%DocumentSubType_NonNegInteger	Non-negative integer

For the %Document_String and %Document_Text primitive types, the values are:

Numeric Value	***Constant Value***	***Description***
—	*none*	No subtype (default).
1	%DocumentSubType_AnyURI	anyURI XML schema data type.
5	%DocumentSubType_gDay	gDay XML schema data type.
6	%DocumentSubType_gMonth	gMonth XML schema data type.
7	%DocumentSubType_gYear	gYear XML schema data type.
8	%DocumentSubType_gYearMonth	gYearMonth XML schema data type.
2	%DocumentSubType_NormString	Normalized string XML schema data type.
4	%DocumentSubType_QName	QName XML schema data type.
3	%DocumentSubType_Token	Token XML schema data type.

This property is read-only.

See Also

Chapter 15, "Document Classes," PrimitiveType, page 746

PrimitiveType

Description

Use this property to return the type for this primitive element as an integer. The values are:

Numeric Value	***Constant Value***	***Description***
11	%Document_Binary	Binary primitive type.
1	%Document_Boolean	Boolean primitive type.
2	%Document_Char	Character primitive type. The Character primitive type allows a single-character string only.
7	%Document_Date	Date primitive type.

Numeric Value	***Constant Value***	***Description***
9	%Document_DateTime	DateTime primitive type.
6	%Document_Decimal	Decimal primitive type.
4	%Document_Integer	Integer primitive type. The Integer primitive type includes subtypes.
3	%Document_String	String primitive type. The String primitive type requires a length. The String primitive type also includes subtypes.
10	%Document_Text	Text primitive type. The Text primitive type is unbounded and does not require a length. The Text primitive type also includes subtypes.
8	%Document_Time	Time primitive type.

This property is read-only.

See Also

Chapter 15, "Document Classes," PrimitiveSubType, page 745

SequenceNumber

Description

Use this property to return the system-assigned sequence number for this element as an integer.

This property is read-only.

Value

Description

Use this property to set or return the value for this primitive element as a string.

This property is read-write.

See Also

Chapter 15, "Document Classes," IsChanged, page 743; Chapter 15, "Document Classes," IsInitialized, page 744 and Chapter 15, "Document Classes," OrigValue, page 745

Compound Class

This section provides an overview of the Compound class and discusses:

- Compound class methods.
- Compound class properties.

The Compound class provides the ability to work with a compound document element.

Compound Class Methods

In this section, the Compound class methods are presented in alphabetical order.

GetParent

Syntax

`GetParent``()`

Description

Use this method to return the parent object of the compound element.

Parameters

None.

Returns

A Collection object or a Compound object.

See Also

Chapter 15, "Document Classes," Collection Class, page 753

Chapter 15, "Document Classes," Compound Class, page 748

GetPath

Syntax

`GetPath``()`

Description

Use this method to return the absolute path to the compound element within the document's structure.

For example, for the Items compound of the PuchaseOrder document, GetPath would return the following absolute path:

```
PurchaseOrder/Item_Collection/Items
```

Parameters

None.

Returns

A string representing the absolute path to the compound element.

GetPropertyByIndex

Syntax

`GetPropertyByIndex`*`(index)`*

Description

Use this method to return a property of the compound element as an object.

Parameters

None.

Parameter	*Description*
index	Specifies the index number of the property as an integer.

Returns

A Collection object, a Compound object, or a Primitive object.

See Also

Chapter 15, "Document Classes," GetPropertyByName, page 750 and Chapter 15, "Document Classes," PropertyCount, page 752

Chapter 15, "Document Classes," Collection Class, page 753

Chapter 15, "Document Classes," Compound Class, page 748

Chapter 15, "Document Classes," Primitive Class, page 741

GetPropertyByName

Syntax

`GetPropertyByName`*`(prop_name)`*

Description

Use this method to return a property of the compound element as an object.

Parameters

Parameter	*Description*
prop_name	Specifies the name of the property as a string.

Returns

A Collection object, a Compound object, or a Primitive object.

See Also

Chapter 15, "Document Classes," GetPropertyByIndex, page 749

Chapter 15, "Document Classes," Collection Class, page 753

Chapter 15, "Document Classes," Compound Class, page 748

Chapter 15, "Document Classes," Primitive Class, page 741

GetUniqueKey

Syntax

`GetUniqueKey()`

Description

Use this method to return the key information for this compound—that is, the package name, document name, and document version.

Parameters

None.

Returns

An array of string with three elements: package name, document name, and document version.

Compound Class Properties

In this section, the Compound class properties are presented in alphabetical order.

ElementType

Description

Use this property to return the type of this document element as an integer constant: %Document_Compound.

This property is read-only.

IsChanged

Description

Use this property to return a Boolean value indicating whether this compound element has been changed.

This property is read-only.

IsInitialized

Description

Use this property to return a Boolean value indicating whether this compound element included initial data when the document was loaded from the XML source.

This property is read-only.

IsRequired

Description

Use this property return a Boolean value indicating whether this compound element is required.

This property is read-only.

Name

Description

Use this property to return the name of this element as a string.

This property is read-only.

PropertyCount

Description

Use this property to return the number of existing properties for this compound element as an integer.

This property is read-only.

See Also

Chapter 15, "Document Classes," GetPropertyByIndex, page 749

SequenceNumber

Description

Use this property to return the system-assigned sequence number for this element as an integer.

This property is read-only.

Collection Class

This section provides an overview of the Collection class and discusses:

- Collection class methods.
- Collection class properties.

The Collection class provides the ability to work with a collection document element.

Collection Class Methods

In this section, the Collection class methods are presented in alphabetical order.

AppendItem

Syntax

`AppendItem`*`(&Elem)`*

Description

Use this method to append an element as the last element of the collection. The item must be instantiated first by using the CreateItem method.

Parameters

Parameter	Description
&Elem	Specifies the element to be appended as an object: a Collection object, a Compound object, or a Primitive object.

Returns

A Boolean value: True if the append was completed successfully, False otherwise.

Example

The following example demonstrates how to use Collection and Compound methods to create, populate, and append a compound item within a collection:

```
Local DocumentKey &DOCKEY;
Local Document &DOC;
Local Compound &COM, &Com_Rdr;
Local Collection &Coll_Rdr;
Local Primitive &PRIM;

&DOCKEY = CreateDocumentKey("FlightStatus", "FlightData", "v1");
&DOC = CreateDocument(&DOCKEY);

&COM = &DOC.DocumentElement;

&Coll_Rdr = &COM.GetPropertyByName("RdrCollection");

&Com_Rdr = &Coll_Rdr.CreateItem();

&PRIM = &Com_Rdr.GetPropertyByName("QE_ACNUMBER");
&PRIM.Value = 105;
&PRIM = &Com_Rdr.GetPropertyByName("QE_AZIMUTH");
&PRIM.Value = "40";
&PRIM = &Com_Rdr.GetPropertyByIndex(3);
&PRIM.Value = "4B";

&nRet = &Coll_Rdr.AppendItem(&Com_Rdr);
```

See Also

Chapter 15, "Document Classes," CreateItem, page 754 and Chapter 15, "Document Classes," CollectionElementType, page 758

CreateItem

Syntax

`CreateItem``()`

Description

Use this method to instantiate an empty object for this collection as defined by the CollectionElementType property.

Parameters

None.

Returns

A Collection object, a Compound object, or a Primitive object as defined by the CollectionElementType property.

See Also

Chapter 15, "Document Classes," AppendItem, page 753; Chapter 15, "Document Classes," InsertItem, page 757 and Chapter 15, "Document Classes," CollectionElementType, page 758

DeleteItem

Syntax

`DeleteItem`*`(index)`*

Description

Use this method to delete the specified element from the collection.

Parameters

Parameter	*Description*
index	Specifies the sequence number of the element to be deleted as an integer.

Returns

A Boolean value: True if the delete was successful, False otherwise.

GetItem

Syntax

`GetItem`*`(index)`*

Description

Use this method to return the specified element from the collection.

Parameters

Parameter	***Description***
index	Specifies the sequence number of the element to be retrieved as an integer.

Returns

A Collection object, a Compound object, or a Primitive object as defined by the CollectionElementType property.

See Also

Chapter 15, "Document Classes," CollectionElementType, page 758

GetParent

Syntax

`GetParent``()`

Description

Use this method to return the parent object of the collection element.

Parameters

None.

Returns

A Collection object or a Compound object.

See Also

Chapter 15, "Document Classes," Collection Class, page 753

Chapter 15, "Document Classes," Compound Class, page 748

GetPath

Syntax

`GetPath``()`

Description

Use this method to return the absolute path to the collection element within the document's structure.

For example, for the Item_Collection collection of the PuchaseOrder document, GetPath would return the following absolute path:

```
PurchaseOrder/Item_Collection
```

Parameters

None.

Returns

A string representing the absolute path to the collection element.

InsertItem

Syntax

`InsertItem``(&Elem,index)`

Description

Use this method to insert an element at the specified location in the collection. The item must be instantiated first by using the CreateItem method.

Parameters

Parameter	*Description*
&Elem	Specifies the element to be inserted as an object: a Collection object, a Compound object, or a Primitive object.
index	Specifies the sequence number for the element to be inserted as an integer.

Returns

A Boolean value : True if the insertion was completed successfully, False otherwise.

Example

For example, for a collection with four items (a, b, c, d), the following call to insert item f would result in a collection with five items (a, f, b, c, d):

```
&ret = &Coll.InsertItem(&f, 2);
```

See Also

Chapter 15, "Document Classes," CreateItem, page 754 and Chapter 15, "Document Classes," CollectionElementType, page 758

Collection Class Properties

In this section, the Collection class properties are presented in alphabetical order.

CollectionElementType

Description

Use this property to return the element type for the collection items as an integer constant. The values are:

Numeric Value	*Constant Value*	*Description*
2	%Document_Collection	A Collection object.
1	%Document_Compound	A compound object.
0	%Document_Primitive	A primitive object.

This property is read-only.

Count

Description

Use this property to return the count of items in the collection as an integer.

This property is read-only.

DefinedMaxOccurs

Description

Use this property to return an integer indicating the maximum number of items in the collection. The integer constant %Document_OccursUnbounded indicates that there is no maximum.

This property is read-only.

DefinedMinOccurs

Description

Use this property to return an integer indicating the minimum number of items in the collection.

This property is read-only.

ElementType

Description

Use this property to return the type of this document element as an integer constant: %Document_Collection.

This property is read-only.

IsChanged

Description

Use this property to return a Boolean value indicating whether this collection element has been changed.

This property is read-only.

IsInitialized

Description

Use this property to return a Boolean value indicating whether this collection included initial data when the document was loaded from the XML source.

This property is read-only.

IsRequired

Description

Use this property return a Boolean value indicating whether this collection element is required.

This property is read-only.

Name

Description

Use this property to return the name of this element as a string.

This property is read-only.

SequenceNumber

Description

Use this property to return the system-assigned sequence number for this element as an integer.

This property is read-only.

Chapter 16

Exception Class

This chapter provides an overview of Exception class and discusses the following topics:

- How exceptions are thrown.
- When to use exceptions.
- Try-Catch sections.
- Data Type for Exception class objects.
- Scope of Exception class objects.
- Exception class built-in function and language constructs.
- Exception class methods.
- Exception class properties.

Understanding Exception Class

An *exception* can be defined as any unusual event that requires special handling in your PeopleCode program. This could include some kind of hardware failure, such as trying to write to a printer that's been turned off, or a software failure, such as trying to divide by zero. You can also detect your own errors, and use the Throw statement to construct an exception.

Exception handling is the processing you initiate when an exception occurs. You can handle errors in PeopleCode using the Catch statement. The places in your code where you want exceptions handled must be enclosed by the Try and End-Try statements.

The Exception class encapsulates exceptional runtime conditions. It can be used with most PeopleCode programs.

Note. The Exception class does *not* work with any of the PeopleSoft APIs, that is, the classes whose objects are declared as type ApiObject. This includes the Tree classes, the Query classes, Search classes, and so on.

The process of error handling can be broken down as follows:

- An error occurs (either a hardware or software error).
- The error is detected and an exception is thrown (either by the system or by your program).
- Your exception handler provides the response.

How Exceptions Are Thrown

Exceptions get thrown either by the runtime system or by your program.

In general, the kinds of errors that cause the runtime system to throw an exception are errors that terminate your PeopleCode program. For example, dividing by zero, referencing a method or property that doesn't exist, or trying to use a method or property on a null object.

To throw your own exceptions, you can use the Throw statement.

When to Use Exceptions

If the error is something that you can easily check, you shouldn't use exceptions. For example, suppose your page had begin and end date fields. In your SavePreChange PeopleCode program, you would want to check to verify that the end date was after the begin date. This kind of error produces incorrect data. It doesn't terminate your PeopleCode program. This is a simple check, one that you don't need to throw an exception for.

Errors that you might want to use exceptions for are the kinds that you are going to check for often. If you do not want to catch a specific exception, but any general exception, then a simple try-catch block is all that is required. Just write code to catch the general Exception object. For example:

```
try
  &RETCODE = GetAttachment(&IB_ATTACH_REC, &SOURCEFILENAME, &SOURCEFILENAME, "PS_⇒
FILEDIR");
catch Exception &err
  &RETCODE = GetAttachment(&IB_ATTACH_REC, &SOURCEFILENAME, "/files/" |⇒
 &SOURCEFILENAME, "PS_SERVDIR");
end-try;
```

However, it might make more sense to have a single exception class for that kind of error, so you'd only have to write your error handling code once. For example, suppose that a function you called was supposed to return an array object that you'd then manipulate using the array class methods and properties. When the function fails and doesn't return an array object, you can either check for null, or check for an exception (as this kind of error terminates your PeopleCode program). If you must often check for nulls, perhaps instead of constantly checking, you could rely on the runtime system throwing exceptions for you.

The following provides a pseudo-code example:

```
import MYAPP:Exception;
/* This subpackage contains all the exception classes for your app */

try
   &MyArray = GetArrayData(&Param1, &Param2, &Param3);
   /* Code to manipulate &MyArray here */

catch ExceptionNull &Ex1
   if &Ex1.MessageSetNumber = 2
      and &Ex1.MessageNumber = 236 Then

   /* This is the message set and message number for &MyArray being Null */

   /* do data error handling */

   end-if;
end-try;
```

Considerations Using Exceptions and Transfer Functions

Using functions that transfer the end user to a new component, such as the DoModal, DoModalComponent, or Transfer functions (in some cases) inside a try block do *not* catch PeopleCode exceptions thrown in the new component. Starting a new component starts a brand new PeopleCode evaluation context. Catches are only caught for exceptions thrown within the *current* component.

In the following code example, the catch statement only catches exceptions thrown in the code *prior to* using the DoModal function, but not any exceptions that are thrown within the new component.

```
/* Set up transaction */
If %CompIntfcName = "" Then
   try
      &oTrans = &g_ERMS_TransactionCollection.GetTransactionByName(RB_EM_⇒
WRK.DESCR);
      &sSearchPage = &oTrans.SearchPage;
      &sSearchRecord = &oTrans.SearchRecord;
      &sSearchTitle = &oTrans.GetSearchPageTitle();
      If Not All(&sSearchPage, &sSearchRecord, &sSearchTitle) Then
         Error (MsgGetText(17834, 8081, "Message Not Found"));
      End-If;
      &c_ERMS_SearchTransaction = &oTrans;

      /* Attempt to transfer to hidden search page with configurable filter */
      &nModalReturn = DoModal(@("Page." | &sSearchPage), &sSearchTitle, - 1, - 1);
   catch Exception &e
      Error (MsgGetText(17834, 8082, "Message Not Found"));
   end-try;
```

See Also

Chapter 6, "Application Classes," Exception Handling, page 220

Try-Catch Sections

The statements that immediately follow the Try statement are called the *protected statements.* These are the only statements that are 'protected' by the catch clauses in the Try-Catch statements. The catch clauses in a Try-Catch statement can be executed only if an exception is thrown by the protected statements. In addition, a catch clause is executed only when handling an exception that matches the type given on the catch. Any exceptions thrown by the catch clauses are not caught by their own Try-Catch statement.

The execution of the Try-Catch statement starts by executing the protected statements. If none of these statements, as well as none of the statements called by them, causes an exception to be thrown, the try-catch is done. The statements in the catch clauses are not executed.

Each catch clause in the Try-Catch section declares a local variable of class Exception or a subclass of Exception. A catch clause local variable has the same scope and lifetime as a local variable declared at the start of the catch clause.

The following is the general order of exception execution and handling in the Try-Catch statements.

1. The exception is thrown.
2. The exception is considered by the first enclosing Try-Catch statement.

   Try-catch statements can be nested.

   This is a dynamic enclosure, that is, tracking back through method and function callers. Any intervening non-Try-Catch statements are skipped.
3. Within this Try-Catch statement, the catch clauses are considered in program text order, that is, first the first catch clause is considered, then the next, and so on to the last. At each consideration, the type (class) of the exception being thrown is compared to the type of the local variable declared by the Catch clause. If the exception being thrown is the same as the Catch type, or is a subtype of it (so the exception could be assigned to the Catch's local variable) the Catch clause matches the exception being thrown.
4. If a matching catch clause is found, that catch clause handles the exception. The exception is considered 'caught', which means that it does not propagate further. Execution proceeds by assigning the thrown object to the catch clause's local variable and executing the statements in the catch clause. When one of the catch clauses handles the exception, (and the statements in the catch clause don't throw any further exceptions), execution continues normally after the Try-Catch statement.

   If a statement in the catch clause throws another exception, that exception begins to propagate from the point of the throw.
5. If a matching catch clause is not found, that is, if no catch clauses can accept the thrown object, then the thrown object is still uncaught. The thrown object is considered by the next dynamically enclosing Try-Catch statement (return to step 3).
6. If a thrown object is not caught by any catch clause in any dynamically enclosing Try-Catch statements, a fatal PeopleCode evaluation error is produced.

Data Type for Exception Class Objects

Exception Class objects are declared as type Exception. For example:

```
Local Exception &Ex1;
```

Scope of Exception Class Objects

Exception objects can be instantiated only from PeopleCode. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

Exception Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateException

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Throw

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Try

Exception Class Methods

In this section, we discuss the Exception class methods, in alphabetical order.

GetSubstitution

Syntax

GetSubstitution*(Index)*

Description

When you create a message in the message catalog, you can specify values in the message text, such as %1, %2, and so on, that are replaced with values by the system at runtime. The value that gets put into the message text value is called the *substitution string*. The GetSubstitution method returns the substitution string in the error message specified by *Index*, with 1 being the first substitution string, 2 being the second, and so on.

Parameters

Parameter	*Description*
Index	Specify the substitution string number that you'd like to return. The first one is 1, the second is 2, and so on. You must specify a valid substitution string number, that is, you can't specify a 4 if there are only three substitution strings.

Returns

The substitution text as a string.

Example

```
If &Ex1.MessageSetNumber = 2 and &Ex1.MessageNumber = 170 Then
   /* Get Sendmail error */
      &String = &Ex1.GetSubstitution(1);
   /* do processing according to type of Sendmail error */
   . . . .
End-if;
```

See Also

Chapter 16, "Exception Class," SetSubstitution, page 767

PeopleTools 8.51 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Message Catalog

Output

Syntax

`Output``()`

Description

Use the Output method if you want to display the message associated with the exception to the user (in an online context), or have it logged (if offline). This is analogous to using the MessageBox function to log a message.

Parameters

None.

Returns

If called in an online context, display the message associated with the exception to the user. If called offline, record the exception for later display or analysis.

Example

```
catch Except2 &Ex2
   &Ex2.Output(); /* tell about it */
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Throw

SetSubstitution

Syntax

SetSubstitution*(Index,String)*

Description

When you create a message in the message catalog, you can specify values in the message text, such as %1, %2, and so on, that are replaced with values by the system at runtime. The value that gets put into the message text value is called the *substitution string.* The SetSubstitution method sets the substitution string in the error message specified by *Index*, with 1 being the first substitution string, 2 being the second, and so on.

Parameters

Parameter	*Description*
Index	Specify the substitution string number that you'd like to replace. The first one is 1, the second is 2, and so on. You must specify a valid substitution string number, that is, you can't specify a 4 if there are only three substitution strings.
String	Specify the text of the substitution string.

Returns

None.

See Also

Chapter 16, "Exception Class," GetSubstitution, page 765

PeopleTools 8.51 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Message Catalog

ToString

Syntax

ToString([*AddContext*])

Description

The ToString method returns the expanded message text associated with the exception in the current user's current language. The string returned is the message text (in the current user's language) with the substitution string replacing the substitution markers (%1 and so on) in the text. It always returns the context information about where the error occurred at the end of the message. To suppress this, specify *AddContext* as false.

Parameters

Parameter	*Description*
AddContext	Specify whether to have the context information about where the error occurred added at the end of the message. This parameter takes a Boolean value. The default is true, that is, to add the context information.

Returns

A string.

See Also

Chapter 16, "Exception Class," Output, page 766

Exception Class Properties

In this section, we discuss the Exception class properties, in alphabetical order.

Context

Description

This property returns a string description of the location of the condition in PeopleCode or other processing. This might contain a stack backtrace at the point of the exceptional condition.

This property is read-only.

Example

This is the "At" part of the error message display. For example, for an exception thrown in Record.Field Event QE_31DIGREC1.QE_31DIGFLD5 FieldChange function EachComp at PeopleCode program counter 671, statement 11 of the source program, the Context would be:

```
At QE_31DIGREC1.QE_31DIGFLD5.FieldChange EachComp   PCPC:671  Statement:11
```

DefaultText

Description

This property sets a string to use as the basic message text for the exception when the message can't be found in the message catalog.

This property is read-write.

MessageNumber

Description

This property is the message number for a message describing the exceptional condition.

This property is read-only.

MessageSetNumber

Description

This property is the message set number for a message describing the exceptional condition.

This property is read-only.

MessageSeverity

Description

This property is the message severity as a string for a message describing the exceptional condition. The message severity is set for the message when it's created in the message catalog. The values are:

Numeric Value	***Constant Value***	***Description***
C	%MsgSeverity_Cancel	Message severity is Cancel.
E	%MsgSeverity_Error	Message severity is Error.
M	%MsgSeverity_Message	Message severity is Message.
W	%MsgSeverity_Warning	Message severity is Warning.

This property is read-only.

StackTrace

Description

This property returns a string with the complete stack trace. This includes the whole context, (that is, current location + called from X + called from. . .)

You can process the returned string by breaking it into pieces using the Split function, and using "Called from" as the value to use for separating the string. For example,

```
Local Array of String &Pieces;

/* get exception */

&Pieces = Split(&E.StackTrace, "Called from:");
```

This property is read-only.

Example

The following is an example of a trace stack.

```
In C (0,0) AEMINITEST.MAIN.GBL.default.1900-01-01.Step02.OnExecute  Name:C  PCPC
:136  Statement:2
Called from:AEMINITEST.MAIN.GBL.default.1900-01-01.Step02.OnExecute  Name:B  Sta
tement:6
Called from:AEMINITEST.MAIN.GBL.default.1900-01-01.Step02.OnExecute  Name:A  Sta
tement:13
Called from:AEMINITEST.MAIN.GBL.default.1900-01-01.Step02.OnExecute  Statement:1
8
```


SubstitutionCount

Description

This property returns the number of substitution strings in the message text as a number.

The number of substitutions comes from the *subslist* of the CreateException function.

This property is read-only.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateException

Chapter 17

Feed Classes

This chapter provides an overview of the feed classes and discusses:

- Importing feed classes
- Feed class
- FeedFactory class
- DataSource class
- DataSourceParameter class
- DataSourceParameterValue class
- DataSourceSetting class
- Utility class
- FeedDoc class
- FeedEntry class
- Additional feed examples

Understanding the Feed Classes

This chapter documents the following feed classes from the PTFP_FEED application package:

- Feed class
- FeedFactory class
- DataSource subpackage:
    - DataSource class
    - DataSourceParameter class
    - DataSourceParameterValue class
    - DataSourceSetting class
- UTILITY subpackage: Utility class

- XML_FEED subpackage
    - FeedDoc class
    - FeedEntry class

The subpackages and classes not documented in this chapter include:

- DataSource subpackage:
    - FeedDataSource
    - GenericDataSource
    - PSQueryDataSource
    - WorklistDataSource
- EXCEPTION subpackage:
    - DuplicateIDException class
    - FeedException class
    - ImmutableException class
    - InitializationException class
    - InvalidMethodException class
    - InvalidTypeException class
    - InvalidValueException class
    - NotAllowedException class
    - NotFoundException class
    - PrivilegeException class
    - PropertyNotSetException class
- Interface subpackage:
    - IBGetFeed class
    - IBGetFeedList class
    - IBGetPrePubFeed class

- UTILITY subpackage
  - Attribute class
  - Authorization class
  - Collection class
  - FeedRequest class
  - HoverMenu class
  - IBOperation class
  - Link class
  - PSComponent class
  - PublishAsRequest class
  - RelatedFeedsRequest class
  - SearchRequest class
- XML_ATOM10 subpackage
  - A10_FeedDoc class
  - A10_FeedEntry class
- XML_OPML subpackage
  - OPMLDoc class
  - OPMLEntry class
  - OPMLFolder class

See Also

PeopleTools 8.51 PeopleBook: Feed Publishing Framework

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker

Importing Feed Classes

The feed classes are application classes, not built-in classes, like Rowset, Field, Record, and so on. Before you can use these classes in your PeopleCode program, you must import them into your program.

An import statement either names a particular application class or imports all the classes in a package.

Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are not made available.

Feed Class

This section provides an overview of the Feed class and discusses:

- Feed class constructor.
- Feed class methods.
- Feed class properties.

The Feed class provides the object representation of a feed. The class provides all the basic services for the feed definition and execution of a feed.

Feed Class Constructor

This section presents the constructor for the Feed class. Use the FeedFactory class to create an instance of the Feed class.

Feed

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.createFeed("feed_ID");
```

See Also

Chapter 17, "Feed Classes," createFeed, page 801

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

Feed Class Methods

In this section, the Feed class methods are presented in alphabetical order.

delete

Syntax

delete()

Description

Use this method to delete a feed definition from the database.

Parameters

None.

Returns

A Boolean value: True if the deletion was successful, False otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local boolean &succeed;
Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DELETION);

If &thisFeed <> Null And
      &thisFeed.Authorized Then
   &succeed = &thisFeed.delete();
End-If;
```

See Also

Chapter 17, "Feed Classes," deleteFeed, page 802

equals

Syntax

equals*(&Object)*

Description

Use this method to evaluate the equivalency of an object with the current feed. Equivalency is established based on the value of two properties: the object's ID and the ObjectType.

Parameters

Parameter	*Description*
&Object	Specifies the object to be compared to the current Feed object.

Returns

A Boolean value: True if the items are equivalent, False otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed1 = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);
Local PTFP_FEED:Feed &thisFeed2 = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);

If &thisFeed1.equals(&thisFeed2) Then
   /* Do some process */
End-If;
```

See Also

Chapter 17, "Feed Classes," ID, page 795 and Chapter 17, "Feed Classes," ObjectType, page 796

execute

Syntax

execute()

Description

Use this method to execute the current feed definition to get a FeedDoc object.

Parameters

None.

Returns

A FeedDoc object.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;
import PTFP_FEED:XML_FEED:FeedDoc;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_EXECUTION);
Local PTFP_FEED:XML_FEED:FeedDoc &thisFeedDoc;

If &thisFeed <> Null Then
   &thisFeed.populatePrefData( Null);
   &thisFeedDoc = &thisFeed.execute();
End-If;
```

See Also

Chapter 17, "Feed Classes," getFeedDoc, page 809

Chapter 17, "Feed Classes," FeedDoc Class, page 902

getAttribute

Syntax

getAttribute*(attribute_ID)*

Description

Use this method to get a feed attribute by ID.

Parameters

Parameter	*Description*
attribute_ID	Specifies the ID of the Attribute object as a string.

Returns

A feed Attribute object.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;
import PTFP_FEED:UTILITY:Attribute;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);
Local PTFP_FEED:UTILITY:Attribute &logo;

If &thisFeed <> Null Then
   &logo = &thisFeed.getAttribute(&thisFeedFactory.Utility.FEEDATTRIBUTE_LOGOURL);
End-If;
```

See Also

Chapter 17, "Feed Classes," FEEDATTRIBUTE_AUTHOR, page 878; Chapter 17, "Feed Classes," FEEDATTRIBUTE_CLOUD, page 878; Chapter 17, "Feed Classes," FEEDATTRIBUTE_COMPLETE, page 878; Chapter 17, "Feed Classes," FEEDATTRIBUTE_CONTRIBUTOR, page 878; Chapter 17, "Feed Classes," FEEDATTRIBUTE_COPYRIGHT, page 879; Chapter 17, "Feed Classes," FEEDATTRIBUTE_EXPIRES, page 879; Chapter 17, "Feed Classes," FEEDATTRIBUTE_ICONURL, page 879; Chapter 17, "Feed Classes," FEEDATTRIBUTE_LOGOURL, page 879; Chapter 17, "Feed Classes," FEEDATTRIBUTE_MANAGINGEDITOR, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_MAXAGE, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_PERSINSTRUCTION, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_RATING, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_SKIPDAYS, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_SKIPHOURS, page 881; Chapter 17, "Feed Classes," FEEDATTRIBUTE_TEXTINPUT, page 881; Chapter 17, "Feed Classes," FEEDATTRIBUTE_TTL, page 881; Chapter 17, "Feed Classes," FEEDATTRIBUTE_WEBMASTER, page 881 and Chapter 17, "Feed Classes," FEEDATTRIBUTE_XSL, page 882

load

Syntax

load()

Description

Use this method to load the feed definition from the database.

Parameters

None.

Returns

A Boolean value: True if the feed definition was loaded successfully, False otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);

If &thisFeed <> Null Then
   &succeed = &thisFeed.load();
End-If;
```

populatePrefData

Syntax

populatePrefData*(&FeedRequest)*

Description

Use this method to populate the administrator- or user-specified data source parameter values from the feed definition or from the feed request if parameters are specified there.

Parameters

Parameter	***Description***
&FeedRequest	Specifies a FeedRequest object.

Returns

None.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;
import PTFP_FEED:XML_FEED:FeedDoc;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_EXECUTION);
Local PTFP_FEED:XML_FEED:FeedDoc &thisFeedDoc;

If &thisFeed <> Null Then
   &thisFeed.populatePrefData( Null);
   &thisFeedDoc = &thisFeed.execute();
End-If;
```

publishToSites

Syntax

publishToSites*(Sites)*

Description

Use this method to publish the feed definition to the specified sites.

Parameters

Parameter	***Description***
Sites	Specifies as an array of string the list of sites to which to publish the feed definition.

Returns

None.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);

If &thisFeed <> Null Then
   &thisFeed.publishToSites(CreateArray("EMPLOYEE", "CUSTOMER"));
End-If;
```

resetFeedAttributes

Syntax

resetFeedAttributes()

Description

Use this method to reset the FeedAttributes collection.

Parameters

None.

Returns

An empty FeedAttributes collection.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;
import PTFP_FEED:UTILITY:Collection;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);
Local PTFP_FEED:UTILITY:Collection &coll;

If &thisFeed <> Null Then
   &coll = &thisFeed.resetFeedAttributes();
End-If;
```

See Also

Chapter 17, "Feed Classes," FeedAttributes, page 791

resetFeedSecurities

Syntax

`resetFeedSecurities``()`

Description

Use this method to reset the FeedSecurities collection.

Parameters

None.

Returns

An empty FeedSecurities collection.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;
import PTFP_FEED:UTILITY:Collection;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);
Local PTFP_FEED:UTILITY:Collection &coll;

If &thisFeed <> Null Then
   &coll = &thisFeed.resetFeedSecurities();
End-If;
```

See Also

Chapter 17, "Feed Classes," FeedSecurities, page 793

save

Syntax

save()

Description

Use this method to save the feed definition to the database.

Parameters

None.

Returns

A Boolean value: True if the save was successful, False otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;
import PTFP_FEED:UTILITY:Attribute;

Local boolean &succeed;
Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);
Local PTFP_FEED:UTILITY:Attribute &logo;

If &thisFeed <> Null Then
   &logo = &thisFeed.setAttribute(&thisFeedFactory.Utility.⇒
FEEDATTRIBUTE_LOGOURL, "", "http://myserver.com/logo.jpg");
   &succeed = &thisFeed.save();
End-If;
```

saveAs

Syntax

saveAs(*new_ID*, *CopyPrefData*)

Description

Use this method to save the feed definition to the database using the given new ID.

Parameters

Parameter	*Description*
new_ID	Specifies the new feed ID as a string.
CopyPrefData	Specifies as a Boolean value whether to copy administrator and user personalization data.

Returns

A new Feed object.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local boolean &succeed;
Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);
Local PTFP_FEED:Feed &newFeed;

If &thisFeed <> Null Then
   &newFeed = &thisFeed.saveAs("new_feed_id", True);
End-If;
```

setAttribute

Syntax

setAttribute(*attribute_ID*,*Value*,*XML*)

Description

Use this method to set the properties of a feed attribute. If the attribute ID does not exist, then a new attribute is created. If both *Value* and *XML* are empty, the attribute is deleted.

Parameters

Parameter	*Description*
attribute_ID	Specifies the ID of the attribute as a string.

Parameter	*Description*
Value	Specifies the value of the attribute as a string. Use *Value* for translatable values such as the copyright and so on.
XML	Specifies the XML value of the attribute as a string. Use *XML* for nontranslatable values or values longer than 254 characters such as a URL.

Returns

An Attribute object.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;
import PTFP_FEED:UTILITY:Attribute;

Local boolean &succeed;
Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);
Local PTFP_FEED:UTILITY:Attribute &logo;

If &thisFeed <> Null Then
   &logo = &thisFeed.setAttribute(&thisFeedFactory.Utility.⇒
FEEDATTRIBUTE_LOGOURL, "", "http://myserver.com/logo.jpg");
   &succeed = &thisFeed.save();
End-If;
```

setDataSourceById

Syntax

setDataSourceById*(DataType_ID)*

Description

Use this method to set the data source for the Feed object by data type ID.

Parameters

Parameter	*Description*
DataType_ID	Specifies the data type ID to set as the data source.

Returns

A DataSource object if successful, null otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;
import PTFP_FEED:DataSource:DataSource;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.createFeed("feed_ID");
Local PTFP_FEED:DataSource:DataSource &thisDS;

&thisDS = &thisFeed.setDataSourceByID("FEED");
```

See Also

Chapter 17, "Feed Classes," DataSource Class, page 815

unpublishFromSites

Syntax

unpublishFromSites*(Sites)*

Description

Use this method to remove the published feed definition from the specified sites.

Parameters

Parameter	***Description***
Sites	Specifies as an array of string the list of sites from which the feed definition is to be removed.

Returns

None.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);

If &thisFeed <> Null Then
   &thisFeed.unpublishFromSites(CreateArray("EMPLOYEE", "CUSTOMER"));
End-If;
```

Feed Class Properties

In this section, the Feed class properties are presented in alphabetical order.

Authorized

Description

This property returns a Boolean value indicating whether the current user is authorized to view the feed definition.

This property is read-only.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local boolean &succeed;
Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DELETION);

If &thisFeed <> Null And
      &thisFeed.Authorized Then
   &succeed = &thisFeed.delete();
End-If;
```

CategoryID

Description

Use this property to set or return the category ID for the feed definition as a string.

This property is read-write.

CreateDTTM

Description

This property returns the date and time the feed definition was created as a datetime value.

This property is read-only.

CreateNode

Description

Use this property to set or return the node on which the feed definition was created, as a string.

This property is read-write.

CreateOprID

Description

This property returns user ID of the user who created the feed definition, as a string.

This property is read-only.

CreatePortal

Description

Use this property to set or return the portal on which the feed definition was created, as a string.

This property is read-write.

DataSource

Description

Use this property to set or return a DataSource object for the feed definition.

This property is read-write.

See Also

Chapter 17, "Feed Classes," DataSource Class, page 815

DataTypeID

Description

This property returns the data type ID for the feed definition's data source, as a string.

This property is read-only.

Description

Description

Use this property to set or return the description of the feed definition as a string.

This property is read-write.

FeedAttributes

Description

Use this property to set or return a FeedAttributes collection.

This property is read-write.

FeedAuthorizationOprID

Description

Use this property to set or return the user ID to be used to execute the Feed object, as a string. This value overrides the current user ID depending on the authorization type.

This property is read-write.

FeedAuthorizationOprPWD

Description

Use this property to set or return the password to be used to execute the Feed object, as a string. This value overrides the password for the current user ID depending on the authorization type.

This property is read-write.

FeedAuthorizationType

Description

Use this property to set or return the authorization type to be used to execute the Feed object, as a string. This value overrides the current authorization type. The values are:

Value	***Description***
&utility.FEEDAUTHTYPE_DEFAULT	The current authenticated user is used to execute the Feed object.
&utility.FEEDAUTHTYPE_ANONYMOUS	The specified user is used to execute the Feed object for anonymous requests.
&utility.FEEDAUTHTYPE_ALL	The specified user is used to execute the Feed object for all requests.

This property is read-write.

FeedCacheTime

Description

This property is not used currently.

This property is read-write.

FeedCacheType

Description

This property is not used currently.

This property is read-write.

FeedContentUrl

Description

This property returns a string representing the content URL to open the content page of the feed.

This property is read-only.

FeedFactory

Description

Use this property to set or return the FeedFactory object used by this Feed object.

This property is read-write.

See Also

Chapter 17, "Feed Classes," FeedFactory Class, page 798

FeedFormat

Description

Use this property to set or return the feed format as a string. The values are:

Value	***Description***
&utility.FEEDFORMAT_ATOM10	Specifies the Atom 1.0 format.

This property is read-write.

FeedSecurities

Description

Use this property to set or return a FeedSecurities collection.

This property is read-write.

FeedSecurityType

Description

Use this property to set or return the security type for the feed as a string. The values are

Value	*Description*
&utility.FEEDSECUTYPE_PUBLIC	Specifies public access to the feed.
&utility.FEEDSECUTYPE_SELECTED	Specifies role or permission list based access to the feed. The role or permission list is stored with the feed definition
&utility.FEEDSECUTYPE_REALTIME	Specifies real-time security in which the DataSource object is used to validate user access when the feed is requested.

This property is read-write.

FeedUrl

Description

This property returns a string representing the feed URL to open the feed document.

This property is read-only.

HasAdminParams

Description

This property returns a Boolean value indicating whether the feed definition has administrator-specified data source parameters.

This property is read-only.

HasUserParams

Description

This property returns a Boolean value indicating whether the feed definition has user-specified data source parameters.

This property is read-only.

IBOperationName

Description

Use this property to set or return a string representing the Integration Broker service operation name that handles requests for this feed.

This property is read-write.

ID

Description

This property returns the ID for the feed definition as a string.

This property is read-only.

LastPubDTTM

Description

This property returns the datetime value at which the data source's execute method was called to publish feed messages to the Integration Broker queues—that is, the last publication date and time for the feed. The initial value for this property is set to 01-01-1900 12:00AM.

This property is not valid for real-time feeds or for scheduled, generic Integration Broker feeds. It is updated for all other types of scheduled feeds.

This property is read-only.

LastUpdDTTM

Description

This property returns the last update date and time for the feed definition, as a datetime value.

This property is read-only.

LastUpdOprID

Description

This property returns the user ID of the user to have last updated the feed definition, as string.

This property is read-only.

NameSpaceID

Description

Use this property to set or return the name space ID for the feed definition, as a string. The default name space ID is based on the feed ID as follows: PTFP_*node_name_feed_ID.*

This property is read-write.

ObjectType

Description

This property returns the object type for the feed definition as a string. For a feed, this is a constant value: Feed.

This property is read-only.

OperatingMode

Description

This property returns the operating mode for the Feed object as a number. The values are:

Numeric Value	***Constant Value***	***Description***
0	&utility.OPERATINGMODE_DEFAULT	A Feed object in this mode allows all operations such as save, delete, and so on.
10	&utility.OPERATINGMODE_EXECUTION	A Feed object in this mode is for execution to get the feed document. Other operations such as save and delete are not allowed.

Numeric Value	Constant Value	Description
11	&utility.OPERATINGMODE_EXECUTION_NOENTRY	A Feed object in this mode is for execution to get a feed document with no feed entries. Otherwise, this mode is the same as the OPERATINGMODE_EXECUTION mode.
100	&utility.OPERATINGMODE_AUTHORIZATION	A Feed object in this mode is for validating user access to the feed. Most other operations such as execute, save, and delete are not allowed.
1000	&utility.OPERATINGMODE_DELETION	A Feed object in this mode is to delete the feed definition. Most other operations such as execute and save are not allowed.

This property is read-only.

OwnerID

Description

Use this property to set or return object owner ID as a string. For example, for delivered feed definitions, the owner ID is set to PT, CPA, and so on.

This property is read-write.

PublishedInSites

Description

This property returns the list of sites in which this feed has been published as an array of string.

This property is read-only.

Title

Description

Use this property to set or return the title of the feed definition as a string.

This property is read-write.

Utility

Description

Use this property to set or return the Utility object used by the Feed object.

This property is read-write.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.getFeed("feed_ID", ⇒
&thisFeedFactory.Utility.OPERATINGMODE_DEFAULT);
Local PTFP_FEED:UTILITY:Utility &utility;

If &thisFeed <> Null Then
   &utility = &thisFeed.Utility;
End-If;
```

See Also

Chapter 17, "Feed Classes," Utility Class, page 853

FeedFactory Class

This section provides an overview of the FeedFactory class and discusses:

- FeedFactory class constructor.
- FeedFactory class methods.
- FeedFactory class properties.

The FeedFactory class provides methods for instantiating objects—such as, Feed objects, DataSource objects, Link objects, and Link collections—rather than instantiating these objects directly. Using the methods of the FeedFactory class avoids creating invalid objects. The FeedFactory class also provides methods to search feed definitions and return the search results as a feed HoverMenu object or OPML file

FeedFactory Class Constructor

This section presents the constructor for the FeedFactory class.

FeedFactory

Example

```
import PTFP_FEED:FeedFactory;

Local PTFP_FEED:FeedFactory &PTFP_FEED_FACTORY = create PTFP_FEED:FeedFactory();
```

See Also

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

FeedFactory Class Methods

In this section, the FeedFactory class methods are presented in alphabetical order.

convertFeedLinksToHoverMenu

Syntax

convertFeedLinksToHoverMenu_(&Links,Flat)_

Description

Use this method to convert a feed Link collection to a HoverMenu object.

Parameters

Parameter	***Description***
&Links	Specifies the feed links as a Link collection.
Flat	A Boolean value that indicates whether to organize links in subfolders based on their categories (False), or into a root folder if there are no categories (True). The default value is False.

Returns

A HoverMenu object if successful.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:UTILITY:SearchRequest;
import PTFP_FEED:UTILITY:Collection;
import PTFP_FEED:UTILITY:HoverMenu;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:UTILITY:Collection &results;
Local PTFP_FEED:UTILITY:SearchRequest &request;
Local PTFP_FEED:UTILITY:HoverMenu &menu;

&request = create PTFP_FEED:UTILITY:SearchRequest("SearchResults");
&request.PortalName = %Portal;
&results = &thisFeedFactory.getFeedLinks(&request);
&menu = &thisFeedFactory.convertFeedLinksToHoverMenu(&results, True);
```

convertFeedLinksToOPML

Syntax

convertFeedLinksToOPML*(&Links,Flat)*

Description

Use this method to convert a feed Link collection to an OPMLDoc object.

Parameters

Parameter	***Description***
&Links	Specifies the feed links as a Link collection.
Flat	A Boolean value that indicates whether to organize links in subfolders based on their categories (False), or into a root folder if there are no categories (True). The default value is False.

Returns

An OPMLDoc object if successful.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:UTILITY:SearchRequest;
import PTFP_FEED:UTILITY:Collection;
import PTFP_FEED:XML_OPML:OPMLDoc;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:UTILITY:Collection &results;
Local PTFP_FEED:UTILITY:SearchRequest &request;
Local PTFP_FEED:XML_OPML:OPMLDoc &opmlDoc;

&request = create PTFP_FEED:UTILITY:SearchRequest("SearchResults");
&request.PortalName = %Portal;
&results = &thisFeedFactory.getFeedLinks(&request);
&opmlDoc = &thisFeedFactory.convertFeedLinksToOPML(&results, True);
```

createFeed

Syntax

createFeed*(feed_ID)*

Description

Use this method to create a new Feed object with the given ID.

Parameters

Parameter	*Description*
feed_ID	Specifies the feed ID of the new feed definition.

Returns

A Feed object if successful, null otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed = &thisFeedFactory.createFeed("feed_ID");
```

See Also

Chapter 17, "Feed Classes," Feed Class, page 776

deleteFeed

Syntax

deleteFeed*(feed_ID)*

Description

Use this method to delete the feed definition.

Parameters

Parameter	*Description*
feed_ID	Specifies the feed definition to be deleted.

Returns

A Boolean value: True if the feed deletion was successful, False otherwise.

Example

```
import PTFP_FEED:FeedFactory;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local boolean &succeed = &thisFeedFactory.deleteFeed("feed_ID");
```

See Also

Chapter 17, "Feed Classes," delete, page 777

genFeedUrl

Syntax

genFeedUrl*(feed_ID,OperationName,SecurityType)*

Description

Use this method to generate the feed URL based on the feed ID and Integration Broker service operation name. If the service operation name is empty, the default service operation, PTFP_GETFEED, is used.

Parameters

Parameter	*Description*
feed_ID	Specifies the ID for the feed definition as a string.
OperationName	Specifies the Integration Broker service operation name as a string. The default service operation is PTFP_GETFEED.
SecurityType	Specifies the security type for the feed as a string.

Returns

A feed URL as a string.

Example

```
import PTFP_FEED:FeedFactory;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local string &url = &thisFeedFactory.genFeedUrl("feed_ID", "PTFP_GETFEED", ⇒
&thisFeedFactory.Utility.FEEDSECUTYPE_PUBLIC);
```

See Also

Chapter 17, "Feed Classes," FEEDSECUTYPE_PUBLIC, page 883; Chapter 17, "Feed Classes," FEEDSECUTYPE_REALTIME, page 884 and Chapter 17, "Feed Classes," FEEDSECUTYPE_SELECTED, page 884

genUniqueFeedId

Syntax

genUniqueFeedId*(seed)*

Description

Use this method to generate an unique feed ID from the specified seed string.

Parameters

Parameter	*Description*
seed	Specifies the seed as a string.

Returns

A feed ID as a string.

Example

```
import PTFP_FEED:FeedFactory;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local string &id = &thisFeedFactory.genUniqueFeedId("feed_ID");
```

getAllPagedFeedLinks

Syntax

getAllPagedFeedLinks*(feed_ID,fromDTTM,allLanguages)*

Description

Use this method to get all the paged feed URLs for the specified scheduled feed definition starting from the specified date and time. If *allLanguages* is set to True, then feed URLs corresponding to all languages are also provided.

This method returns an array of paged feed URLs; each URL represents a specific page (message segment) of the paged feed.

Note. This method is supported for scheduled feeds only.

See Also

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

Parameters

Parameter	*Description*
feed_ID	Specifies the ID for the scheduled feed definition as a string.
fromDTTM	Specifies the start date and time as a datetime value.
allLanguages	A Boolean value that indicates whether to provide paged feed URLs for all languages.

Returns

An array of string.

Example

```
import PTFP_FEED:FeedFactory;

Local PTFP_FEED:FeedFactory &feedFactory_inst;
Local datetime &fromDttm;
Local array of string &feedLinks;

&feedFactory_inst = create PTFP_FEED:FeedFactory();
&fromDttm = DateTimeValue("01/01/1900 00:00:00");
&feedLinks = &feedFactory_inst.getAllPagedFeedLinks("ADMN_USER_PROFILE", ⇒
&fromDttm, True);
```

See Also

Chapter 17, "Feed Classes," getRelativePageLinkForFeed, page 812

getCategory

Syntax

getCategory*(category_ID,ActiveOnly)*

Description

Use this method to retrieve all of the category information for a category ID. The information is returned as an array of string with four elements.

Parameters

Parameter	Description
category_ID	Specifies the category ID as a string.
ActiveOnly	Indicates as a Boolean value whether to retrieve the description only if the category is active.

Returns

An array of string with four elements:

- Category ID
- Short description
- Long description
- Status

Example

```
import PTFP_FEED:FeedFactory;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local array of string &catagories = &thisFeedFactory.getCategory⇒
("category_ID", True);
```

getDataSource

Syntax

getDataSource*(DataType_ID)*

Description

Use this method to create a DataSource object with the given data type ID.

Parameters

None.

Parameter	Description
DataType_ID	Specifies the data type ID for the data source, as a string.

Returns

A DataSource object if successful, null otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:DataSource:DataSource;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:DataSource:DataSource &thisDS;

&thisDS = &thisFeedFactory.getDataSource("FEED");
```

See Also

Chapter 17, "Feed Classes," DataSource Class Constructor, page 815

getFeed

Syntax

getFeed(*feed_ID*,*mode*)

Description

Use this method to create a Feed object with the given feed ID.

Parameters

Parameter	***Description***
feed_ID	Specifies the feed ID as a string.
mode	Specifies the operating mode for the feed as a number. See the values below.

The values for the *mode* parameter are:

Numeric Value	***Constant Value***	***Description***
0	&utility.OPERATINGMODE_DEFAULT	A Feed object in this mode allows all operations such as save, delete, and so on.

Numeric Value	Constant Value	Description
10	&utility.OPERATINGMODE_EXECUTION	A Feed object in this mode is for execution to get the feed document. Other operations such as save and delete are not allowed.
11	&utility.OPERATINGMODE_EXECUTION_NOENTRY	A Feed object in this mode is for execution to get a feed document with just an empty feed header (that is, without feed entries). Otherwise, this mode is the same as the OPERATINGMODE_EXECUTION mode.
100	&utility.OPERATINGMODE_AUTHORIZATION	A Feed object in this mode is for validating user access to the feed. Most other operations such as execute, save, and delete are not allowed.
1000	&utility.OPERATINGMODE_DELETION	A Feed object in this mode is to delete the feed definition. Most other operations such as execute and save are not allowed.

Returns

A Feed object if successful, null otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:Feed;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:Feed &thisFeed;

&thisFeed = &thisFeedFactory.getFeed("feed_ID", &thisFeedFactory.Utility.⇒
OPERATINGMODE_DEFAULT);
```

See Also

Chapter 17, "Feed Classes," Feed Class Constructor, page 776

Chapter 17, "Feed Classes," OPERATINGMODE_AUTHORIZATION, page 890; Chapter 17, "Feed Classes," OPERATINGMODE_DEFAULT, page 890; Chapter 17, "Feed Classes," OPERATINGMODE_DELETION, page 890; Chapter 17, "Feed Classes," OPERATINGMODE_EXECUTION, page 891 and Chapter 17, "Feed Classes," OPERATINGMODE_EXECUTION_NOENTRY, page 891

getFeedDoc

Syntax

`getFeedDoc`*`(&feedRequest)`*

Description

Use this method to get the feed document of the feed specified by the FeedRequest object.

Parameters

Parameter	***Description***
&feedRequest	Specifies the feed request as a FeedRequest object.

Returns

A FeedDoc object if successful, null otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:XML_FEED:FeedDoc;
import PTFP_FEED:UTILITY:FeedRequest;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:UTILITY:FeedRequest &request;
Local PTFP_FEED:XML_FEED:FeedDoc &thisFeedDoc;

&request = create PTFP_FEED:UTILITY:FeedRequest("FeedRequest");
&request.FeedID = "feed_ID";
&thisFeedDoc = &thisFeedFactory.getFeedDoc(&request);
```

See Also

Chapter 17, "Feed Classes," execute, page 778

Chapter 17, "Feed Classes," FeedDoc Class, page 902

getFeedLink

Syntax

getFeedLink*(feed_ID)*

Description

Use this method to create a Link object for the specified feed ID.

Parameters

Parameter	***Description***
feed_ID	Specifies the feed ID as a string.

Returns

A Link object if successful, null otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:UTILITY:Link;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:UTILITY:Link &thisFeedLink;

&thisFeedLink = &thisFeedFactory.getFeedLink("feed_ID");
```

getFeedLinks

Syntax

getFeedLinks*(&searchRequest)*

Description

Use this method to search feed definitions based on user permissions and the search criteria specified in the SearchRequest object. The method returns a collection of feed Link objects. The list is sorted by the feed definition's last updated time by default.

Parameters

Parameter	Description
&searchRequest	Specifies the search criteria as a SearchRequest object.

Returns

A Link collection.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:UTILITY:SearchRequest;
import PTFP_FEED:UTILITY:Collection;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:UTILITY:Collection &results;
Local PTFP_FEED:UTILITY:SearchRequest &request;

&request = create PTFP_FEED:UTILITY:SearchRequest("SearchResults");
&request.PortalName = %Portal;
&results = &thisFeedFactory.getFeedLinks(&request);
```

getRelatedFeedsHoverMenu

Syntax

getRelatedFeedsHoverMenu*(&Requests)*

Description

Use this method to create a related feeds HoverMenu object from the specified FeedRequest objects.

Parameters

Parameter	Description
&Requests	Specifies the related feeds requests as an array of RelatedFeedsRequest objects.

Returns

A HoverMenu object.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:UTILITY:RelatedFeedsRequest;
import PTFP_FEED:UTILITY:HoverMenu;
import PTFP_FEED:UTILITY:Collection;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:UTILITY:Collection &results;
Local PTFP_FEED:UTILITY:RelatedFeedsRequest &request;
Local PTFP_FEED:UTILITY:HoverMenu &menu;

&request = create PTFP_FEED:UTILITY:RelatedFeedsRequest("FEED");
&request.DataTypeID = "FEED";
&menu = &thisFeedFactory.getRelatedFeedsHoverMenu(CreateArray(&request));
```

getRelativePageLinkForFeed

Syntax

getRelativePageLinkForFeed*(feed_ID,currentFeedURL, linkOption,fromDTTM)*

Description

Use this method to get the specified relative paged feed URL (first, previous, next or last) for the current paged feed URL.

This method returns a paged feed URL as a string.

Note. This method is supported for scheduled feeds only.

See Also

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

Parameters

Parameter	***Description***
feed_ID	Specifies the ID for the scheduled feed definition as a string.
currentFeedURL	Specifies the paged feed URL for the current page as a string.
linkOption	Specifies which page link as a string. See the values below.
fromDTTM	Specifies the start date and time as a datetime value.

Value	*Description*
&utlity.LINKTYPE_FIRST	Indicates the first page of the paged feed.
&utlity.LINKTYPE_PREVIOUS	Indicates the previous page of the paged feed.
&utlity.LINKTYPE_NEXT	Indicates the next page of the paged feed.
&utlity.LINKTYPE_LAST	Indicates the last page of the paged feed.

Returns

A string.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:UTILITY:Utility;
import PTFP_FEED:UTILITY:Link;

Local PTFP_FEED:FeedFactory &feedFactory_inst;
Local PTFP_FEED:UTILITY:Utility &utility;
Local PTFP_FEED:UTILITY:Link &link;

Local datetime &fromDttm;
Local string &feedLink, &nextFeedLink, &prevFeedLink, &firstFeedLink, ⇒
&lastFeedLink;

&feedFactory_inst = create PTFP_FEED:FeedFactory();
&utility = &feedFactory_inst.Utility;
&fromDttm = DateTimeValue("01/01/1900 00:00:00");
&link = &feedFactory_inst.getFeedLink("ADMN_USER_PROFILE");
If All(&link) Then
   &feedLink = &link.Href;

   &nextFeedLink = &feedFactory_inst.getRelativePageLinkForFeed⇒
("ADMN_USER_PROFILE", &feedLink, &utility.LINKTYPE_NEXT, &fromDttm);
   &prevFeedLink = &feedFactory_inst.getRelativePageLinkForFeed⇒
("ADMN_USER_PROFILE", &feedLink, &utility.LINKTYPE_PREVIOUS, &fromDttm);
   &firstFeedLink = &feedFactory_inst.getRelativePageLinkForFeed⇒
("ADMN_USER_PROFILE", &feedLink, &utility.LINKTYPE_FIRST, &fromDttm);
   &lastFeedLink = &feedFactory_inst.getRelativePageLinkForFeed⇒
("ADMN_USER_PROFILE", &feedLink, &utility.LINKTYPE_LAST, &fromDttm);
End-If;
```

See Also

Chapter 17, "Feed Classes," getAllPagedFeedLinks, page 804

FeedFactory Class Properties

In this section, the FeedFactory class properties are presented in alphabetical order.

DataSources

Description

This property returns a collection of all data source definitions (a collection of DataSource objects).

This property is read-only.

See Also

Chapter 17, "Feed Classes," DataSource Class, page 815

Feeds

Description

This property returns a collection of all feed definitions (a collection of Feed objects).

This property is read-only.

See Also

Chapter 17, "Feed Classes," Feed Class, page 776

Utility

Description

This property returns the Utility object used by this FeedFactory object.

This property is read-only.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:UTILITY:Utility &utility;

&utility = &thisFeedFactory.Utility;
```

See Also

Chapter 17, "Feed Classes," Utility Class, page 853

DataSource Class

This section provides an overview of the DataSource class and discusses:

- DataSource class constructor.
- DataSource class methods.
- DataSource class properties.

The DataSource class is an abstract base class and should not be used directly. All feed data sources should extend this class.

DataSource Class Constructor

This section presents the constructor for the DataSource class. Use the FeedFactory class to create an instance of the DataSource class.

DataSource

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:DataSource:DataSource;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:DataSource:DataSource &thisDS = &thisFeedFactory.⇒
getDataSource("DS_ID");
```


See Also

Chapter 17, "Feed Classes," getDataSource, page 806

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

DataSource Class Methods

In this section, the DataSource class methods are presented in alphabetical order.

addParameter

Syntax

addParameter*(DSparam_ID,Value)*

Description

Use this method to create a DataSourceParameter object with the given ID and value.

Parameters

Parameter	*Description*
DSparam_ID	Specifies the ID of the data source parameter as a string.
Value	Specifies the value of the data source parameter as a string.

Returns

A DataSourceParameter object if successful, false otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:DataSource:DataSource;
import PTFP_FEED:DataSource:DataSourceParameter;
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:DataSource:DataSource &thisDS = &thisFeedFactory.⇒
getDataSource("DS_ID");
Local PTFP_FEED:DataSource:DataSourceParameter &thisDSP;
If (&thisDS <> Null) Then
   Local PTFP_FEED:UTILITY:Utility &utility = &thisDS.Utility;

   &thisDSP = &thisDS.addParameter(&utility.DSPARAMETER_MAXROW, "0");
   If (&thisDSP <> Null) Then
      &thisDSP.Name = &thisDSP.ID;
      &thisDSP.Description = "Description";
      &thisDSP.FieldType = &utility.FIELDTYPE_NUMBER;
      &thisDSP.DefaultValue = "10";
      &thisDSP.Value = &thisDSP.DefaultValue;
      &thisDSP.Required = True;
   End-If;
End-If;
```

See Also

Chapter 17, "Feed Classes," DataSourceParameter Class, page 831

Chapter 17, "Feed Classes," DSPARAMETER_MAXROW, page 876; Chapter 17, "Feed Classes," DSPARAMETER_SF_MAXMINUTES, page 876 and Chapter 17, "Feed Classes," DSPARAMETER_SF_PAGING, page 876

equals

Syntax

equals*(&Object)*

Description

Use this method to evaluate the equivalency of an object with the current data source. Equivalency is established based on the value of two properties: the object's ID and the ObjectType.

Parameters

Parameter	*Description*
&Object	Specifies the object to be compared to the current DataSource object.

Returns

A Boolean value: True if the items are equivalent, False otherwise.

See Also

Chapter 17, "Feed Classes," ID, page 828 and Chapter 17, "Feed Classes," ObjectType, page 829

getAttributeById

Syntax

`getAttributeById``(attribute_ID)`

Description

Use this method to return the default feed attribute value of this data type by ID.

Parameters

Parameter	*Description*
attribute_ID	Specifies the ID of the attribute as a string.

Returns

An Attribute object if successful, null otherwise.

See Also

Chapter 17, "Feed Classes," FEEDATTRIBUTE_AUTHOR, page 878; Chapter 17, "Feed Classes," FEEDATTRIBUTE_CLOUD, page 878; Chapter 17, "Feed Classes," FEEDATTRIBUTE_COMPLETE, page 878; Chapter 17, "Feed Classes," FEEDATTRIBUTE_CONTRIBUTOR, page 878; Chapter 17, "Feed Classes," FEEDATTRIBUTE_COPYRIGHT, page 879; Chapter 17, "Feed Classes," FEEDATTRIBUTE_EXPIRES, page 879; Chapter 17, "Feed Classes," FEEDATTRIBUTE_ICONURL, page 879; Chapter 17, "Feed Classes," FEEDATTRIBUTE_LOGOURL, page 879; Chapter 17, "Feed Classes," FEEDATTRIBUTE_MANAGINGEDITOR, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_MAXAGE, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_PERSINSTRUCTION, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_RATING, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_SKIPDAYS, page 880; Chapter 17, "Feed Classes," FEEDATTRIBUTE_SKIPHOURS, page 881; Chapter 17, "Feed Classes," FEEDATTRIBUTE_TEXTINPUT, page 881; Chapter 17, "Feed Classes," FEEDATTRIBUTE_TTL, page 881; Chapter 17, "Feed Classes," FEEDATTRIBUTE_WEBMASTER, page 881 and Chapter 17, "Feed Classes," FEEDATTRIBUTE_XSL, page 882

getContentUrl

Syntax

getContentUrl()

Description

Use this method to return the content URL of the feed as a string.

Parameters

None.

Returns

The content URL of the feed as a string.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:DataSource:DataSource;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:DataSource:DataSource &thisDS = &thisFeedFactory.⇒
getDataSource("DS_ID");
Local string &FeedContentUrl = "";
If (&thisDS <> Null) Then
   &FeedContentUrl = &thisDS.getContentUrl();
End-If;
```

getDataSecurity

Syntax

getDataSecurity()

Description

Use this method to return the role- and permission list-based data security as a collection of Authorization objects.

Parameters

None.

Returns

Null if the data security type is public, otherwise a collection of Authorization objects. Each Authorization object represents a role or permission list that has access to the data.

getParameterById

Syntax

getParameterById(*DSP_ID*)

Description

Use this method to return the DataSourceParameter object with the given ID.

Parameters

Parameter	*Description*
DSP_ID	Specifies the ID of the data source parameter as a string.

Returns

A DataSourceParameter object, null if the data source parameter with the specified ID does not exist.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:DataSource:DataSource;
import PTFP_FEED:DataSource:DataSourceParameter;
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:DataSource:DataSource &thisDS = &thisFeedFactory.⇒
getDataSource("DS_ID");
Local PTFP_FEED:DataSource:DataSourceParameter &thisDSP;
If (&thisDS <> Null) Then
   Local PTFP_FEED:UTILITY:Utility &utility = &thisDS.Utility;

   &thisDSP = &thisDS.getParameterById(&utility.DSPARAMETER_MAXROW);
   /* Do some processing with the data source parameter here */
End-If;
```


See Also

Chapter 17, "Feed Classes," DataSourceParameter Class, page 831

Chapter 17, "Feed Classes," DSPARAMETER_MAXROW, page 876; Chapter 17, "Feed Classes," DSPARAMETER_SF_MAXMINUTES, page 876 and Chapter 17, "Feed Classes," DSPARAMETER_SF_PAGING, page 876

getParameterDetail

Syntax

getParameterDetail()

Description

Use this method to return the detailed explanation text of all data source parameters for the specific combination of data source settings.

Parameters

None.

Returns

A string with the detailed explanation text in HTML.

getSettingById

Syntax

getSettingById*(DSS_ID)*

Description

Use this method to return a DataSourceSetting object for the given ID.

Parameters

Parameter	*Description*
DSS_ID	Specifies the ID of the data source setting as a string.

Returns

A DataSourceSetting object if successful, null if the data source setting with the specified ID does not exist.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:DataSource:DataSource;
import PTFP_FEED:DataSource:DataSourceSetting;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:DataSource:DataSource &thisDS = &thisFeedFactory.⇒
getDataSource("DS_ID");
Local PTFP_FEED:DataSource:DataSourceSetting &thisDSS;
If (&thisDS <> Null) Then
   &thisDSS = &thisDS.getSettingById("DSS_ID");
   /* Do some processing with the data source setting here */
End-If;
```

See Also

Chapter 17, "Feed Classes," DataSourceSetting Class, page 845

getSettingDetail

Syntax

getSettingDetail()

Description

Use this method to return the detailed explanation text of all data source settings.

Parameters

None.

Returns

A string with the detailed explanation text in HTML.

isCurrentUserAdmin

Syntax

isCurrentUserAdmin()

Description

Use this method to determine whether the current user has administrator access to the feed data.

Parameters

None.

Returns

A Boolean value: True if the user has administrator access, False otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:DataSource:DataSource;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:DataSource:DataSource &thisDS = &thisFeedFactory.⇒
getDataSource("DS_ID");
Local boolean &authorized = False;
If (&thisDS <> Null) Then
   &authorized = &thisDS.isCurrentUserAdmin();
   /* Do some processing here */
End-If;
```

isCurrentUserAuthorized

Syntax

isCurrentUserAuthorized()

Description

Use this method to determine whether the current user is authorized to view the feed data.

Parameters

None.

Returns

A Boolean value: True if the user is authorized to view the feed data, False otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:DataSource:DataSource;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:DataSource:DataSource &thisDS = &thisFeedFactory.⇒
getDataSource("DS_ID");
Local boolean &authorized = False;
If (&thisDS <> Null) Then
   &authorized = &thisDS.isCurrentUserAuthorized();
   /* Do some processing here */
End-If;
```

onDelete

Syntax

onDelete()

Description

Use this method to update related data prior to deleting a feed definition. Raise an exception to stop the deleting action.

Parameters

None.

Returns

None.

onSave

Syntax

onSave()

Description

Use this method to update related data after a feed definition has been saved.

Parameters

None.

resetParameters

Syntax

resetParameters()

Description

Use this method to reset the Parameters collection.

Parameters

None.

Returns

An empty DataSourceParameter collection.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:DataSource:DataSource;
import PTFP_FEED:UTILITY:Collection;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:DataSource:DataSource &thisDS = &thisFeedFactory.⇒
getDataSource("DS_ID");
Local PTFP_FEED:UTILITY:Collection &thisList;

If (&thisDS <> Null) Then
   &thisList = &thisDS.resetParameters();
End-If;
```

See Also

Chapter 17, "Feed Classes," Parameters, page 829

DataSource Class Properties

In this section, the DataSource class properties are presented in alphabetical order.

AdminPersonalizationPage

Description

Use this property to set or return a PSComponent object representing a data type-specific administrator personalization page for entering administrator-specified data source parameter values.

This property is read-write.

AllowCustomParameters

Description

This property indicates whether to allow user add or delete data source parameters, as a Boolean value. The default false, custom parameters are not allowed.

This property is read-only.

AllowRealTimeFeedSecurity

Description

This property indicates whether to allow real-time feed security, as a Boolean value. The default false, real-time feed security is not allowed.

This property is read-only.

DataSourceType

Description

This property returns the data source type as a string. The default value is a constant: BaseDataSource.

This property is read-only.

DefaultFeedAttributes

Description

Use this property to set or return the default feed attributes of this data type as an Attribute collection.

This property is read-write.

DefaultIBOperationName

Description

This property returns a string representing the name of the default service operation that handles feed requests of this data type.

This property is read-only.

Description

Description

Use this property to set or return the description of the data type as a string.

This property is read-write.

HasParameters

Description

This property whether the data source has parameters, as a Boolean value.

This property is read-only.

HasSettings

Description

This property whether the data source has settings, as a Boolean value.

This property is read-only.

IBOperations

Description

Use this property to set or return an IBOperation collection with the service operation names that could be used to handle feed requests of this feed type.

This property is read-write.

ID

Description

This property returns the ID for the data type as a string.

This property is read-only.

LongDescription

Description

Use this property to set or return the long description of the data type as a string.

This property is read-write.

ObjectType

Description

This property returns the object type for the data source as a string. For a data source, this is a constant value: DataSource.

This property is read-only.

Parameters

Description

This property returns the DataSoureParameter collection of this data source.

This property is read-only.

ParametersCompleted

Description

This property indicates whether all required data source parameters have values, as a Boolean value.

This property is read-only.

Parent

Description

Use this property to set or return the Feed object that use this DataSource object to collect data.

This property is read-write.

See Also

Chapter 17, "Feed Classes," Feed Class, page 776

PortalSpecificPersonalization

Description

This property returns a Boolean value indicating whether the personalization data is portal specific. The default is False, personalization data is not portal specific.

This property is read-only.

Settings

Description

This property returns the DataSourceSetting collection.

This property is read-only.

See Also

Chapter 17, "Feed Classes," DataSourceSetting Class, page 845

SettingsCompleted

Description

This property returns a Boolean value indicating whether all required data source settings are selected.

This property is read-only.

UserPersonalizationPage

Description

Use this property to set or return a PSComponent object representing a data type-specific user personalization page for entering user specified data source parameter values.

This property is read-write.

Utility

Description

Use this property to set or return the Utility object that used by this DataSource object. By default, the parent's (that is, the feed's) Utility object is used.

This property is read-write.

See Also

Chapter 17, "Feed Classes," Utility Class, page 853

DataSourceParameter Class

This section provides an overview of the DataSourceParameter class and discusses:

- DataSourceParameter class constructor.
- DataSourceParameter class methods.
- DataSourceParameter class properties.

The DataSourceParameter class represent a data source parameter that used by the DataSource object to refine the feed data selection.

DataSourceParameter Class Constructor

This section presents the constructor for the DataSourceParameter class.

DataSourceParameter

Example

```
import PTFP_FEED:DataSource:DataSourceParameter;

Local PTFP_FEED:DataSource:DataSourceParameter &DSP = create PTFP_FEED:DataSource:⇒
DataSourceParameter("DSP_ID", &Parent_DS);
```


See Also

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

DataSourceParameter Class Methods

In this section, the DataSourceParameter class methods are presented in alphabetical order.

addUserValue

Syntax

addUserValue_(DSPValue_ID,Value)_

Description

Use this method to add a user value to this data source parameter.

Parameters

Parameter	*Description*
DSPValue_ID	Specifies the ID of the user value as a string.
Value	Specifies the value of the user value as a string.

Returns

A DataSourceParameterValue object if successful, false otherwise.

See Also

Chapter 17, "Feed Classes," DataSourceParameterValue Class Constructor, page 842

clone

Syntax

clone()

Description

Use this method to make a copy of this DataSourceParameter object.

Parameters

None.

Returns

A DataSourceParameter object exactly matching the current object.

equals

Syntax

equals*(&Object)*

Description

Use this method to evaluate the equivalency of an object with the current data source parameter. Equivalency is established based on the value of two properties: the object's ID and the ObjectType.

Parameters

Parameter	***Description***
&Object	Specifies the object to be compared to the current DataSourceParameter object.

Returns

A Boolean value: True if the items are equivalent, False otherwise.

See Also

Chapter 17, "Feed Classes," ID, page 838 and Chapter 17, "Feed Classes," ObjectType, page 838

resetUserValues

Syntax

resetUserValues()

Description

Use this method to reset the UserValues collection.

Parameters

None.

Returns

An empty UserValues collection if successful, null otherwise.

See Also

Chapter 17, "Feed Classes," UserValues, page 840

setRangeFromFieldTranslates

Syntax

setRangeFromFieldTranslates*(FieldName)*

Description

Use this method to auto-populate the range values based on the translate values for the specified field.

Parameters

Parameter	***Description***
FieldName	Specifies the name of the field as a string.

Returns

None.

validateValue

Syntax

`validateValue``(Value,CheckPrompt)`

Description

Use this method to validate the value according to its field type, the edit type, and the usage type. Date or date time values are translated to the standard format as a string. Exceptions or errors are raised for invalid values.

Parameters

Parameter	***Description***
Value	Specifies the value to be validated as a string.
CheckPrompt	Specifies a Boolean value indicating whether the value should be checked against the valid values list of this data source parameter. The type of check depends on the edit type of this data source parameter, the valid values list could be a user-specified values list, translate values of a field, or values in a prompt table. When *CheckPrompt* is True, check the value; when False, check the data type of the value only.

Returns

A string with the validated value.

DataSourceParameter Class Properties

In this section, the DataSourceParameter class properties are presented in alphabetical order.

AllowChangesToRequired

Description

Use this property to set or return a Boolean value indicating whether it is allowed to change the required flag of this parameter.

This property is read-write.

DefaultValue

Description

Use this property to set or return a string representing the default value for this data source parameter.

This property is read-write.

DefaultValueForDisplay

Description

This property returns a string representing the default value for a data source parameter for display purposes. This is the same as the DefaultValue property except that a date value is in the user-preferred format.

This property is read-only.

Description

Description

Use this property to set or return the description of the data source parameter as a string.

This property is read-write.

EditType

Description

Use this property to set or return the table edit type of the data source parameter, as a number. The values are:

Numeric Value	***Constant Value***	***Description***
1	&utility.EDITTYPE_NOTABLEEDIT	This is the default edit type. A valid values list can be specified using the UserValues collection.
2	&utility.EDITTYPE_PROMPTTABLE	This data source parameter takes value from a prompt table. Only a SQL table or SQL view can be used as the prompt table.
3	&utility.EDITTYPE_TRANSLATETABLE	This data source parameter only takes values specific in the Range array of this data source parameter
4	&utility.EDITTYPE_YESNO	This data source parameter only takes a Yes or No value.

This property is read-write.

EvaluatedValue

Description

This property returns a string representing the evaluated value of the data source parameter. This is the same as the Value property except that a system variable is evaluated to its current runtime value.

This property is read-only.

FieldType

Description

Use this property to set or return the field type as a number. The values are:

Numeric Value	***Constant Value***	***Description***
0	&utility.FIELDTYPE_CHARACTER	Character field
1	&utility.FIELDTYPE_LONGCHARACTER	Long character field
2	&utility.FIELDTYPE_NUMBER	Number field
3	&utility.FIELDTYPE_SIGNEDNUMBER	Signed number field
4	&utility.FIELDTYPE_DATE	Date field

Numeric Value	*Constant Value*	*Description*
5	&utility.FIELDTYPE_TIME	Time field
6	&utility.FIELDTYPE_DATETIME	Datetime field

This property is read-write.

ID

Description

This property returns the ID for this data source parameter as a string.

This property is read-only.

Name

Description

Use this property to set or return the name of this data source parameter as a string.

This property is read-write.

ObjectType

Description

This property returns the object type for the data source parameter as a string. For a DataSourceParameter object, this is a constant value: DataSourceParameter.

This property is read-only.

Parent

Description

Use this property to set or return a pointer to the DataSource object to which this parameter belongs.

This property is read-write.

See Also

Chapter 17, "Feed Classes," DataSource Class, page 815

PromptTable

Description

Use this property to set or return the name of the prompt table as a string. Only a SQL table or SQL view should be used as the prompt table.

This property is read-write.

Range

Description

Use this property to set or return the list of valid values as an array of array of string.

This property is read-write.

Required

Description

Use this property to set or return whether this data source parameter is required, as a Boolean value.

This property is read-write.

SkipValidityChecks

Description

Use this property to set or return whether to skip validity checks of the values for this data source parameter, as a Boolean value.

This property is read-write.

UsageType

Description

Use this property to set or return the usage type for this data source parameter as a string. The values are:

Value	***Description***
&utility.USAGETYPE_FIXED	This parameter takes a fixed value.
&utility.USAGETYPE_NOTUSED	This parameter is not used.
&utility.USAGETYPE_SYSVAR	This parameter gets its value from a system variable.
&utility.USAGETYPE_USERSPECIFIED	The value of this parameter can be specified by the user.
&utility.USAGETYPE_INTERNAL	This parameter is for internal use.
&utility.USAGETYPE_ADMINSPECIFIED	The value of this parameter can be changed only by users having administrator access to the data.

This property is read-write.

UserValues

Description

This property returns the valid values (that is, the UserValues collection).

This property is read-only.

Utility

Description

Use this property to set or return the Utility object used by the DataSourceParameter object. By default, the parent's (that is, the data source's) Utility object is used.

This property is read-write.

Example

```
import PTFP_FEED:DataSource:DataSourceParameter;
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:DataSource:DataSourceParameter &thisDSP;
Local PTFP_FEED:UTILITY:Utility &utility;

&thisDSP = create PTFP_FEED:DataSource:DataSourceParameter("dsp_ID", Null);
&utility = &thisDSP.Utility;
&thisDSP.Name = &thisDSP.ID;
&thisDSP.Description = MsgGetText(219, 3005, "Message Not Found - Max Entries");
&thisDSP.FieldType = &utility.FIELDTYPE_NUMBER;
&thisDSP.DefaultValue = "10";
&thisDSP.Value = &thisDSP.DefaultValue;
&thisDSP.Required = True;
&thisDSP.EditType = &utility.EDITTYPE_NOTABLEEDIT;
&thisDSP.PromptTable = ""; /* Should only use SQL Table or View */
&thisDSP.Range = Null; /* Only set the range if the edit type is translatable */
&thisDSP.UsageType = &utility.USAGETYPE_FIXED;
&thisDSP.UserValues = Null;
&thisDSP.AllowChangesToRequired = False;
```

See Also

Chapter 17, "Feed Classes," Utility Class, page 853

Value

Description

Use this property to set or return the value of the data source parameter as a string.

This property is read-write.

ValueForDisplay

Description

This property returns a string representing the value for a data source parameter for display purposes. This is the same as the Value property except that a date value is in the user-preferred format.

This property is read-only.

DataSourceParameterValue Class

This section provides an overview of the DataSourceParameterValue class and discusses:

- DataSourceParameterValue class constructor.
- DataSourceParameterValue class methods.
- DataSourceParameterValue class properties.

The DataSourceParameterValue class represents a valid value for a data source parameter.

DataSourceParameterValue Class Constructor

This section presents the constructor for the DataSourceParameterValue class.

DataSourceParameterValue

Example

```
Local PTFP_FEED:DataSource:DataSourceParameterValue &DSP_Value = create PTFP_FEED:⇒
DataSource:DataSourceParameterValue("ID", &Parent_DSP);
```

See Also

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

DataSourceParameterValue Class Methods

In this section, the DataSourceParameterValue class methods are presented in alphabetical order.

clone

Syntax

clone()

Description

Use this method to make a copy of this DataSourceParameterValue object.

Parameters

None.

Returns

A DataSourceParameterValue object exactly matching the current object.

equals

Syntax

`equals``(&Object)`

Description

Use this method to evaluate the equivalency of an object with the current data source parameter value. Equivalency is established based on the value of two properties: the object's ID and the ObjectType.

Parameters

Parameter	***Description***
&Object	Specifies the object to be compared to the current DataSourceParameterValue object.

Returns

A Boolean value: True if the items are equivalent, False otherwise.

See Also

Chapter 17, "Feed Classes," ID, page 844 and Chapter 17, "Feed Classes," ObjectType, page 844

DataSourceParameterValue Class Properties

In this section, the DataSourceParameterValue class properties are presented in alphabetical order.

Description

Description

Use this property to set or return the description of this data source parameter value as a string.

This property is read-write.

ID

Description

This property returns the ID for this data source parameter value as a string.

This property is read-only.

Name

Description

Use this property to set or return the name of this data source parameter value as a string.

This property is read-write.

ObjectType

Description

This property returns the object type for the data source parameter value as a string. For a DataSourceParameterValue object, this is a constant value: DataSourceParameterValue.

This property is read-only.

OrderNumber

Description

Use this property to set or return the order number for this data source parameter value as a number.

This property is read-write.

Parent

Description

Use this property to set or return a pointer to the DataSourceParameter object to which this value belongs.

This property is read-write.

See Also

Chapter 17, "Feed Classes," DataSourceParameter Class, page 831

Value

Description

Use this property to set or return the value of the data source parameter value as a string.

This property is read-write.

DataSourceSetting Class

This section provides an overview of the DataSourceSetting class and discusses:

- DataSourceSetting class constructor.
- DataSourceSetting class methods.
- DataSourceSetting class properties.

The DataSourceSetting class represents a data source setting that is used to uniquely define a feed's data source of a given type.

DataSourceSetting Class Constructor

This section presents the constructor for the DataSourceSetting class.

DataSourceSetting

Example

```
import PTFP_FEED:DataSource:DataSourceSetting;

Local PTFP_FEED:DataSource:DataSourceSetting &this_DSS = create PTFP_FEED:⇒
DataSource:DataSourceSetting("DSS_ID", &DS);
```

See Also

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

DataSourceSetting Class Methods

In this section, the DataSourceSetting class methods are presented in alphabetical order.

clone

Syntax

clone()

Description

Use this method to make a copy of this DataSourceSetting object.

Parameters

None.

Returns

A DataSourceSetting object exactly matching the current object.

equals

Syntax

equals*(&Object)*

Description

Use this method to evaluate the equivalency of an object with the current data source setting. Equivalency is established based on the value of two properties: the object's ID and the ObjectType.

Parameters

Parameter	*Description*
&Object	Specifies the object to be compared to the current DataSourceSetting object.

Returns

A Boolean value: True if the items are equivalent, False otherwise.

See Also

Chapter 17, "Feed Classes," ID, page 850 and Chapter 17, "Feed Classes," ObjectType, page 850

setRangeFromFieldTranslates

Syntax

setRangeFromFieldTranslates*(FieldName)*

Description

Use this method to auto-populate the range values based on the valid translate values for the field.

Parameters

Parameter	*Description*
FieldName	Specifies the name of the field as a string.

Returns

None.

DataSourceSetting Class Properties

In this section, the DataSourceSetting class properties are presented in alphabetical order.

DefaultValue

Description

Use this property to set or return a string representing the default value for a data source setting.

This property is read-write.

DisplayOnly

Description

Use this property to set or return a Boolean value indicating whether the Value property is displayed as display only.

This property is read-write.

EditType

Description

Use this property to set or return the table edit type of the data source setting, as a number. The values are:

Numeric Value	*Constant Value*	*Description*
1	&utility.EDITTYPE_NOTABLEEDIT	This is the default edit type. A valid values list can be specified using the UserValues collection.
2	&utility.EDITTYPE_PROMPTTABLE	This data source parameter takes value from a prompt table. Only a SQL table or SQL view can be used as the prompt table.

Numeric Value	*Constant Value*	*Description*
3	&utility.EDITTYPE_TRANSLATETABLE	This data source parameter only takes values specific in the Range array of this data source parameter
4	&utility.EDITTYPE_YESNO	This data source parameter only takes a Yes or No value.

This property is read-write.

Enabled

Description

Use this property to set or return a Boolean value indicating whether the setting is enabled.

This property is read-write.

FieldType

Description

Use this property to set or return the field type as a number. The values are:

Numeric Value	*Constant Value*	*Description*
0	&utility.FIELDTYPE_CHARACTER	Character field
1	&utility.FIELDTYPE_LONGCHARACTER	Long character field
2	&utility.FIELDTYPE_NUMBER	Number field
3	&utility.FIELDTYPE_SIGNEDNUMBER	Signed number field
4	&utility.FIELDTYPE_DATE	Date field
5	&utility.FIELDTYPE_TIME	Time field
6	&utility.FIELDTYPE_DATETIME	Datetime field

This property is read-write.

ID

Description

This property returns the ID for the data source setting as a string.

This property is read-only.

LongName

Description

Use this property to set or return the label of the name as a string.

This property is read-write.

Name

Description

Use this property to set or return the name of the data source setting as a string.

This property is read-write.

ObjectType

Description

This property returns the object type for the data source setting as a string. For a DataSourceSetting object, this is a constant value: DataSourceSetting.

This property is read-only.

Parent

Description

Use this property to set or return a pointer to the DataSource object to which this setting belongs.

This property is read-write.

See Also

Chapter 17, "Feed Classes," DataSource Class, page 815

PromptTable

Description

Use this property to set or return the name of the prompt table as a string. Only a SQL table or SQL view should be used.

This property is read-write.

Range

Description

Use this property to set or return the list of valid values as an array of array of string.

This property is read-write.

RefreshOnChange

Description

Use this property to set or return a Boolean value indicating whether to call the parent DataSource object's processSettingsChange method when the value changes.

This property is read-write.

RelatedFieldValue

Description

Use this property to set or return the related display field value as a string.

This property is read-write.

Required

Description

Use this property to set or return a Boolean value indicating whether this data source setting is required.

This property is read-write.

ShowRelatedField

Description

Use this property to set or return a Boolean value indicating whether to show the related display field.

This property is read-write.

Utility

Description

Use this property to set or return the Utility object used by the DataSourceSetting object. By default, the parent's (that is, the data source's) Utility object is used.

This property is read-write.

Example

```
import PTFP_FEED:DataSource:DataSourceSetting;
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:DataSource:DataSourceSetting &thisDSS;
Local PTFP_FEED:UTILITY:Utility &utility;

&thisDSS = create PTFP_FEED:DataSource:DataSourceSetting("dss_ID", Null);
&utility = &thisDSS.Utility;
&thisDSS.Name = &thisDSS.ID;
&thisDSS.LongName = "Data Type Name";
&thisDSS.FieldType = &utility.FIELDTYPE_CHARACTER;
&thisDSS.DefaultValue = "";
&thisDSS.RelatedFieldValue = "";
&thisDSS.EditType = &utility.EDITTYPE_PROMPTTABLE;
&thisDSS.PromptTable = Record.PTFP_DTYPE_PVW; /* Only use SQL Table or View */
&thisDSS.Range = Null; /* Only set the range if the edit type is translatable */
&thisDSS.Required = True;
&thisDSS.Enabled = True;
&thisDSS.Visible = True;
&thisDSS.DisplayOnly = False;
&thisDSS.ShowRelatedField = True;
&thisDSS.RefreshOnChange = True;
```


See Also

Chapter 17, "Feed Classes," Utility Class, page 853

Value

Description

Use this property to set or return the value of the data source setting as a string.

This property is read-write.

Visible

Description

Use this property to set or return a Boolean value indicating whether to show the Value field when it's enabled.

This property is read-write.

Utility Class

This section provides an overview of the Utility class and discusses:

- Utility class constructor.
- Utility class methods.
- Utility class properties.

The Utility class provides methods and constants used by other classes of the Feed Publishing Framework

Utility Class Constructor

This section presents the constructor for the Utility class.

Utility

Example

```
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:UTILITY:Utility &utility = create PTFP_FEED:UTILITY:Utility();
```

See Also

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

Utility Class Methods

In this section, the Utility class methods are presented in alphabetical order.

dateStringToUserPref

Syntax

dateStringToUserPref*(DateString)*

Description

Use this method to transform a date string from the "yyyy-MM-dd" format to the user-preferred date format.

Parameters

Parameter	***Description***
DateString	Specifies the date string to be transformed.

Returns

A string containing the date in the user-preferred date format.

datetimeToString

Syntax

`datetimeToString`*`(Datetime)`*

Description

Use this method to transform a datetime value to a string in "yyyy-MM-dd HH:mm:ss" format.

Parameters

Parameter	*Description*
Datetime	Specifies the datetime value to be transformed.

Returns

A string in "yyyy-MM-dd HH:mm:ss" format or an empty string if the datetime value is empty.

dateToString

Syntax

`dateToString`*`(Date)`*

Description

Use this method to transform a date value to a string in "yyyy-MM-dd" format.

Parameters

Parameter	*Description*
Date	Specifies the date value to be transformed.

Returns

A string in "yyyy-MM-dd" format or an empty string if the date value is empty.

decodeXML

Syntax

`decodeXML`*`(String)`*

Description

Use this method to decode the following encoded XML characters in the source string:

Encoded Character	***Decoded Character***
<	< (less than)
>	> (greater than)
&	& (ampersand)
'	' (apostrophe)
"	" (quotation mark)

Parameters

Parameter	***Description***
String	Specifies the source XML string with encoded characters to be decoded.

Returns

A string containing the source XML string with encoded characters decoded.

Example

```
For &i = 1 To &elementList.Len
   &valueList = CreateArrayRept("", &pChildTags.Len);
   For &j = 1 To &pChildTags.Len
      &thisValue = %This.getNodeValue(&elementList [&i], &pChildTags [&j]);
      &valueList [&j] = %This.decodeXML(&thisValue);
   End-For;
   &return_value.Push(&valueList);
End-For;
```


See Also

Chapter 17, "Feed Classes," encodeXML, page 857 and Chapter 17, "Feed Classes," split2D, page 871

encodeXML

Syntax

`encodeXML`*`(String)`*

Description

Use this method to encode the following special characters in the source string as encoded XML characters:

Character	*Encoded Character*
< (less than)	<
> (greater than)	>
& (ampersand)	&
' (apostrophe)	'
" (quotation mark)	"

Parameters

Parameter	*Description*
String	Specifies the source string with special characters to be encoded.

Returns

A string containing the source string with the special characters encoded.

Example

```
If (&pValues [&i] <> Null) And
      (&pChildTags <> Null) Then
   /* get the children */
   For &j = 1 To &pValues [&i].Len
      If &j > &pChildTags.Len Then
         Break;
      End-If;

      &thisElement = &thisElement | %This.setNodeValue(%This.encodeXML⇒
(&pValues [&i][&j]), &pChildTags [&j]);
   End-For;
End-If;
```

See Also

Chapter 17, "Feed Classes," decodeXML, page 856 and Chapter 17, "Feed Classes," join2D, page 866

evaluateSysVar

Syntax

evaluateSysVar*(SysVar)*

Description

Use this method to evaluate and return the value of a system variable.

Parameters

Parameter	*Description*
SysVar	Specifies the system variable to be evaluated as a string.

Returns

A string representation of the evaluated system variable, or the system variable name if it cannot be evaluated.

genNameSpaceID

Syntax

`genNameSpaceID``(`*`NameSpace_ID`*`)`

Description

Use this method to replace special characters in the original ID string with "_", so that it only contains "A–Z", "0–9" and "_".

Parameters

Parameter	*Description*
NameSpace_ID	Specifies the name space ID to be converted as a string.

Returns

A string containing the generated name space ID using only "A–Z", "0–9" and "_".

getExceptionText

Syntax

`getExceptionText``(`*`&Exception`*`)`

Description

Use this method to return the exception error message.

Parameters

Parameter	*Description*
&Exception	Specifies the exception as an Exception object.

Returns

The exception error message as a string.

getFeedDoc

Syntax

getFeedDoc(*feed_ID*,*Format*,*&Attributes*)

Description

Use this method to create a FeedDoc object of the given feed format.

Parameters

Parameter	***Description***
feed_ID	Specifies the ID of the feed definition as a string.
Format	Specifies the feed format as a string. The values are:

Value	***Description***
&utility.FEEDFORMAT_ATOM10	An Atom 1.0 format feed.

Parameter	***Description***
&Attributes	Specifies the feed-level properties as an Attribute collection.

Returns

A FeedDoc object of the given feed format with properties filled in from the Attribute collection.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:XML_FEED:FeedDoc;
import PTFP_FEED:XML_FEED:FeedEntry;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:XML_FEED:FeedDoc &this_FeedDoc = &thisFeedFactory.Utility.⇒
getFeedDoc("feed_ID", &thisFeedFactory.Utility.FEEDFORMAT_ATOM10, Null);
Local PTFP_FEED:XML_FEED:FeedEntry &this_Entry = &this_FeedDoc.⇒
addEntry("entry_ID");
```

See Also

Chapter 17, "Feed Classes," FeedDoc Class, page 902

getFeedMimeType

Syntax

`getFeedMimeType`*`(Format)`*

Description

Use this method to return the MIME type of the given feed format.

Parameters

Parameter	***Description***
Format	Specifies the feed format as a string. The values are:

Value	***Description***
&utility.FEEDFORMAT_ATOM10	An Atom 1.0 format feed.

Returns

A string containing the MIME type as follows:

Feed Format Constant	***MIME Type Constant***	***MIME Type***
&utility.FEEDFORMAT_ATOM10	MIMETYPE_ATOM	application/atom+xml
undefined	MIMETYPE_XML	application/xml

getFieldTranslates

Syntax

`getFieldTranslates`*`(FieldName)`*

Description

Use this method to return the valid translate values of a field.

Parameters

Parameter	***Description***
FieldName	Specifies the field name for which to retrieve translate values, as a string.

Returns

An array of array of string.

getNodeValue

Syntax

`getNodeValue`*`(String, Tag)`*

Description

Use this method to extract the value from the first element enclosed by the given tag in the source string.

Parameters

Parameter	***Description***
String	Specifies the string to check.
Tag	Specifies the tag to search for as a string.

Returns

A string containing the value from the first element enclosed by the specified tag.

Example

```
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:UTILITY:Utility &utility = create PTFP_FEED:UTILITY:Utility();

&str = "<div><p>value1</p><p>value2</p></div>";

&str1 = &utility.getNodeValue(&str, "p");

/* &str1="value1" */
```

See Also

Chapter 17, "Feed Classes," setNodeValue, page 868; Chapter 17, "Feed Classes," split, page 870 and Chapter 17, "Feed Classes," split2D, page 871

getUserDateFormat

Syntax

getUserDateFormat()

Description

Use this method to construct and return the date format string based on the user's personalization settings.

Parameters

None.

Returns

A string containing the format mask representing the user's preferred date format—for example: MM-dd-yyyy.

getUserDatetimeFormat

Syntax

getUserDatetimeFormat()

Description

Use this method to construct and return the datetime format string based on the user's personalization settings.

Parameters

None.

Returns

A string containing the format mask representing the user's preferred datetime format—for example: MM-dd-yyyy hh:mm a.

getUserInfo

Syntax

`getUserInfo`*`(user_ID)`*

Description

Use this method to return the user name and email address of the given user ID.

Parameters

Parameter	*Description*
user_ID	Specifies the user ID for which to retrieve the user information, as a string.

Returns

An array of string containing two elements:

- User name (or the user ID if this is empty).
- Email address (or an empty string if this is empty).

httpStringToDatetime

Syntax

`httpStringToDatetime`*`(string)`*

Description

Use this method to convert an HTTP date/time string in the "dow, dd mmm yyyy hh:mm:ss GMT" format to a datetime value.

Parameters

Parameter	Description
string	Specifies the HTTP date/time string in the "dow, dd mmm yyyy hh:mm:ss GMT" format.

Returns

A datetime value.

join

Syntax

`join``(&Array,Tag)`

Description

Use this method to concatenate the string array together enclosing each string element in the specified tag.

Parameters

Parameter	Description
&Array	Specifies the array of string to be concatenated together.
Tag	Specifies the tag to enclose each string element.

Returns

A string containing the concatenation of each of the string elements enclosed by the specified tag.

Example

```
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:UTILITY:Utility &utility = create PTFP_FEED:UTILITY:Utility();

&str = &utility.join(CreateArray("value1", "value2"), "p");

/* &str="<p>value1</p><p>value2</p>" */
```

See Also

Chapter 17, "Feed Classes," setNodeValue, page 868 and Chapter 17, "Feed Classes," split, page 870

join2D

Syntax

join2D(*&Array*, *Tag*, *ChildTags*)

Description

Use this method to concatenate the array of array of string together enclosing each element in the specified tag. Each child element is XML encoded and enclosed in the child tag.

Parameters

Parameter	*Description*
&Array	Specifies the array of array of string to be concatenated together.
Tag	Specifies the tag to enclose each string element, as a string.
ChildTags	Specifies the tags to enclose each child element, as an array of string.

Returns

A string containing the concatenation of each of the string elements enclosed by the specified tag. Each child element is XML encoded and enclosed in the child tag.

Example

```
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:UTILITY:Utility &utility = create PTFP_FEED:UTILITY:Utility();

&str = &utility.join2D(CreateArray(CreateArray("value1", "value2")), "div", ⇒
CreateArray("b", "i"));

/* &str="<div><b>value1</b><i>value2</i></div>" */
```

See Also

Chapter 17, "Feed Classes," encodeXML, page 857; Chapter 17, "Feed Classes," setNodeValue, page 868 and Chapter 17, "Feed Classes," split, page 870

setMessageHeadersAndMimeType

Syntax

setMessageHeadersAndMimeType*(&Message,&feeddoc, &FeedRequest)*

Description

Use this method to return the specified Message object with the MIME type set, and populated with the specified FeedDoc. If the feed is an incremental feed, then incremental feed information is set as connector properties for the message.

Parameters

Parameter	*Description*
&Message	Specifies the Message object to be returned.
&feeddoc	Specifies the feed document to be returned specified as an XmlDoc object.
&FeedRequest	Specifies the feed as a FeedRequest object.

Returns

A Message object.

See Also

Chapter 17, "Feed Classes," httpStringToDatetime, page 864

Chapter 17, "Feed Classes," Implementing a Real-Time Feed Request Handler, page 930

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Incremental Feeds

setNodeValue

Syntax

setNodeValue(*Value*, *Tag*)

Description

Use this method to form an element using the given value and tag name.

Parameters

Parameter	***Description***
Value	Specifies the string value to enclose in the tags.
Tag	Specifies the tag to enclose the value within, as a string.

Returns

A string element containing the value enclosed within the given tag.

Example

```
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:UTILITY:Utility &utility = create PTFP_FEED:UTILITY:Utility();

&str1 = &utility.setNodeValue("value", "p");

/* &str="<p>value</p>" */
```

See Also

Chapter 17, "Feed Classes," getNodeValue, page 862; Chapter 17, "Feed Classes," join, page 865 and Chapter 17, "Feed Classes," join2D, page 866

showException

Syntax

`showException`*`(Exception)`*

Description

Use this method to raise an error when the component variable &PTFP_THROWPAGELETEXCEPTIONS is false, otherwise throw the exception.

Parameters

Parameter	***Description***
Exception	Specifies the exception as an Exception.

Returns

None.

showInvalidValueException

Syntax

`showInvalidValueException`*`(Name,Value)`*

Description

Use this method to show an invalid value exception using the showException method.

Parameters

Parameter	***Description***
Name	Specifies the name of the property as a string.
Value	Specifies the value that is invalid as a string.

Returns

None

See Also

Chapter 17, "Feed Classes," showException, page 869

split

Syntax

split()

Description

Use this method to extract the elements from the string that are enclosed in the specified tag.

Parameters

Parameter	*Description*
String	Specifies the string to check.
Tag	Specifies the tag to search for as a string.

Returns

An array of string containing the string elements enclosed by the specified tag.

Example

```
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:UTILITY:Utility &utility = create PTFP_FEED:UTILITY:Utility();

&str = "<p>value1</p><p>value2</p>";

&arr = &utility.split(&str, "p");

/* &arr[1]="value1", &arr[2]="value2" */
```


See Also

Chapter 17, "Feed Classes," getNodeValue, page 862 and Chapter 17, "Feed Classes," join, page 865

split2D

Syntax

split2D()

Description

Use this method to extract the elements from the string that are enclosed in the specified tag. Child elements enclosed by the child tags are also extracted. All elements are XML decoded.

Parameters

Parameter	*Description*
String	Specifies the string to check.
Tag	Specifies the tag that encloses each string element, as a string. This tag can be an empty string.
ChildTags	Specifies the tags that enclose each child element, as an array of string.

Returns

An array of array of string containing the extracted string elements enclosed by the specified tags. Child elements are enclosed by the child tags. All elements are XML decoded.

Example

```
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:UTILITY:Utility &utility = create PTFP_FEED:UTILITY:Utility();

&str = "<div><b>value1</b><i>value2</i></div>";

&arr = &utility.split2D(&str, "div", CreateArray("b", "i"));

/* &arr[1][1]="value1", &arr[1][2]="value2" */
```

See Also

decodeXML, getNodeValue, join2D, split

Chapter 17, "Feed Classes," decodeXML, page 856; Chapter 17, "Feed Classes," getNodeValue, page 862; Chapter 17, "Feed Classes," join2D, page 866 and Chapter 17, "Feed Classes," split, page 870

stringToDate

Syntax

`stringToDate`*`(DateString)`*

Description

Use this method to convert a date string in the "yyyy-MM-dd" format to a date.

Parameters

Parameter	***Description***
DateString	Specifies the string to be converted to a date.

Returns

A date.

stringToDatetime

Syntax

`stringToDatetime`*`(DatetimeString)`*

Description

Use this method to convert a date string in the "yyyy-MM-dd HH:mm:ss" format to a datetime.

Parameters

Parameter	Description
DatetimeString	Specifies the string to be converted to a datetime.

Returns

A datetime.

validateSysVar

Syntax

`validateSysVar`*`(SysVar)`*

Description

Use this method to return the valid, correctly capitalized name of a system variable.

Parameters

Parameter	Description
SysVar	Specifies the system variable name to be validated as a string.

Returns

A string containing the validated and correctly capitalized system variable name, INVALID SYSVAR if the system variable does not exist.

Example

In the following example, %AuthenticationToken is returned as the validated system variable name.

```
import PTFP_FEED:UTILITY:Utility;

Local PTFP_FEED:UTILITY:Utility &utility = create PTFP_FEED:UTILITY:Utility();

&str = &utility.validateSysVar("%AUTHENTICATIONtoken");

/* &str="%AUTHENTICATIONTOKEN" */
```


viewStringAsAttachment

Syntax

`viewStringAsAttachment``(String,FileName,ViewAttachment)`

Description

Use this method to transform the given string into an attachment file that is sent to the browser.

Parameters

Parameter	***Description***
String	Specifies the specifies the contents of the attachment file as a string.
FileName	Specifies the attachment file name as a string.
ViewAttachment	Specifies the as a Boolean value whether the attachment is to be viewed directly or detached for download.

Returns

None.

Utility Class Properties

In this section, the Utility class properties are presented in alphabetical order.

Note. Properties in this class with capitalized names—such as, ATTACHMENT_URL, AUTHTYPE_PERM, and so on—are constants.

Some properties of the Utility class—such as, QUERYPARAMETER_FEEDID , QUERYPARAMETER_FEEDFORMAT, and so on—are referred to as query parameters. Typically, these query parameters are used in the implementation of a feed request handler. An example of a feed request handler that uses query parameters is provided at the end of this chapter.

See Also

Chapter 17, "Feed Classes," Implementing a Real-Time Feed Request Handler, page 930

ATTACHMENT_URL

Description

Use this property to return the attachment location from the PTFP_DOCINDB URL definition, as a string.

This property is read-only.

AUTHTYPE_PERM

Description

Use this property to return the value that represents the permission list-based authentication type, as a string.

This property is read-only.

AUTHTYPE_ROLE

Description

Use this property to return the value that represents the role-based authentication type, as a string.

This property is read-only.

DSPARAMETER_INCREMENTAL

Description

Use this property to return a string representing the name of the incremental feed data source parameter.

This property is read-only.

See Also

Chapter 17, "Feed Classes," INCREMENTALOPTION_NO, page 887 and Chapter 17, "Feed Classes," INCREMENTALOPTION_YES, page 887

DSPARAMETER_MAXROW

Description

Use this property to return a string representing the name of the maximum rows data source parameter for scheduled feeds.

Note. Use either the SF_MAXROWOPTION_ALLMSGS property or the SF_MAXROWOPTION_LATESTMSG property to return the *value* of the maximum rows data source parameter for scheduled feeds.

This property is read-only.

See Also

Chapter 17, "Feed Classes," SF_MAXROWOPTION_ALLMSGS, page 896 and Chapter 17, "Feed Classes," SF_MAXROWOPTION_LATESTMSG, page 896

DSPARAMETER_SF_MAXMINUTES

Description

Use this property to return a string representing the name of the maximum minutes data source parameter for scheduled feeds.

Note. Use the SF_MAXMINUTES_ALLMSGS property to return the *value* of the maximum minutes data source parameter for scheduled feeds.

This property is read-only.

See Also

Chapter 17, "Feed Classes," SF_MAXMINUTES_ALLMSGS, page 895

DSPARAMETER_SF_PAGING

Description

Use this property to return a string representing the name of the paging data source parameter for scheduled feeds.

Note. Use either the SF_PAGINGOPTION_NOPAGING property or the SF_PAGINGOPTION_SEGMENTED property to return the *value* of the paging data source parameter for scheduled feeds.

This property is read-only.

See Also

Chapter 17, "Feed Classes," SF_PAGINGOPTION_NOPAGING, page 897 and Chapter 17, "Feed Classes," SF_PAGINGOPTION_SEGMENTED, page 897

EDITTYPE_NOTABLEEDIT

Description

Use this property to return a number representing the field edit type as no prompt table edit.

This property is read-only.

EDITTYPE_PROMPTTABLE

Description

Use this property to return a number representing the field edit type as prompt table edit.

This property is read-only.

EDITTYPE_TRANSLATETABLE

Description

Use this property to return a number representing the field edit type as translate table edit.

This property is read-only.

EDITTYPE_YESNO

Description

Use this property to return a number representing the field edit type as a Yes/No edit.

This property is read-only.

FEEDATTRIBUTE_AUTHOR

Description

Use this property to return the AUTHOR feed attribute name as a string.

This property is read-only.

See Also

Chapter 17, "Feed Classes," XMLCHILDELEMENTS_PERSON, page 899

FEEDATTRIBUTE_CLOUD

Description

Use this property to return the CLOUD feed attribute name as a string.

This property is read-only.

See Also

Chapter 17, "Feed Classes," XMLCHILDELEMENTS_CLOUD, page 899

FEEDATTRIBUTE_COMPLETE

Description

Use this property to return the COMPLETE feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_CONTRIBUTOR

Description

Use this property to return the CONTRIBUTOR feed attribute name as a string.

This property is read-only.

See Also

Chapter 17, "Feed Classes," XMLCHILDELEMENTS_PERSON, page 899

FEEDATTRIBUTE_COPYRIGHT

Description

Use this property to return the COPYRIGHT feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_EXPIRES

Description

Use this property to return the EXPIRES feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_ICONURL

Description

Use this property to return the ICONURL feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_LOGOURL

Description

Use this property to return the LOGOURL feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_MANAGINGEDITOR

Description

Use this property to return the MANAGINGEDITOR feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_MAXAGE

Description

Use this property to return the MAXAGE feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_PERSINSTRUCTION

Description

Use this property to return the PERSINSTRUCTION feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_RATING

Description

Use this property to return the RATING feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_SKIPDAYS

Description

Use this property to return the SKIPDAYS feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_SKIPHOURS

Description

Use this property to return the SKIPHOURS feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_TEXTINPUT

Description

Use this property to return the TEXTINPUT feed attribute name as a string.

This property is read-only.

See Also

Chapter 17, "Feed Classes," XMLCHILDELEMENTS_TEXTINPUT, page 900

FEEDATTRIBUTE_TTL

Description

Use this property to return the TTL feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_WEBMASTER

Description

Use this property to return the WEBMASTER feed attribute name as a string.

This property is read-only.

FEEDATTRIBUTE_XSL

Description

Use this property to return the XSL feed attribute name as a string.

This property is read-only.

FEEDAUTHTYPE_ALL

Description

Use this property to return a string representing the feed authorization type as all.

This property is read-only.

FEEDAUTHTYPE_ANONYMOUS

Description

Use this property to return a string representing the feed authorization type as anonymous.

This property is read-only.

FEEDAUTHTYPE_DEFAULT

Description

Use this property to return a string representing the feed authorization type as default.

This property is read-only.

FEEDCACHETYPE_NONE

Description

Use this property to return a string representing the feed cache type as none.

This property is read-only.

FEEDCACHETYPE_PRIVATE

Description

Use this property to return a string representing the feed cache type as private.

This property is read-only.

FEEDCACHETYPE_PUBLIC

Description

Use this property to return a string representing the feed cache type as public.

This property is read-only.

FEEDCACHETYPE_ROLE

Description

Use this property to return a string representing the feed cache type as role-based.

This property is read-only.

FEEDFORMAT_ATOM10

Description

Use this property to return a string representing the feed format as Atom 1.0.

This property is read-only.

FEEDSECUTYPE_PUBLIC

Description

Use this property to return a string representing the feed security type as public access.

This property is read-only.

FEEDSECUTYPE_REALTIME

Description

Use this property to return a string representing the feed security type as real-time security.

This property is read-only.

FEEDSECUTYPE_SELECTED

Description

Use this property to return a string representing the feed security type as select security access.

This property is read-only.

FEEDTYPE_DYNAMIC

Description

Use this property to return a string representing the feed type as real time (formerly known as dynamic).

This property is read-only.

FEEDTYPE_PREPUBLISHED

Description

Use this property to return a string representing the feed type as scheduled (formerly known as prepublished).

This property is read-only.

FIELDTYPE_CHARACTER

Description

Use this property to return a number representing the field type as character.

This property is read-only.

FIELDTYPE_DATE

Description

Use this property to return a number representing the field type as date.

This property is read-only.

FIELDTYPE_DATETIME

Description

Use this property to return a number representing the field type as datetime.

This property is read-only.

FIELDTYPE_LONGCHARACTER

Description

Use this property to return a number representing the field type as long character.

This property is read-only.

FIELDTYPE_NUMBER

Description

Use this property to return a number representing the field type as number.

This property is read-only.

FIELDTYPE_SIGNEDNUMBER

Description

Use this property to return a number representing the field type as signed number.

This property is read-only.

FIELDTYPE_TIME

Description

Use this property to return a number representing the field type as time.

This property is read-only.

IBSOTYPE_ASYNC

Description

Use this property to return a string representing the Integration Broker service operation type as asynchronous.

This property is read-only.

IBSOTYPE_SYNC

Description

Use this property to return a string representing the Integration Broker service operation type as synchronous.

This property is read-only.

IBSOTYPE_UNKNOWN

Description

Use this property to return a string representing the Integration Broker service operation type as unknown.

This property is read-only.

ICONURL_FEED_A

Description

Use this property to return the active feed icon URL for PTFP_FEED_ACTIVE, as a string.

This property is read-only.

ICONURL_FEED_IA

Description

Use this property to return the inactive feed icon URL for PTFP_FEED_INACTIVE, as a string.

This property is read-only.

INCREMENTALOPTION_NO

Description

Use this property to return an integer indicating that the feed is not an incremental feed.

This property is read-only.

See Also

Chapter 17, "Feed Classes," DSPARAMETER_INCREMENTAL, page 875

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Incremental Feeds

INCREMENTALOPTION_YES

Description

Use this property to return an integer indicating that the feed is an incremental feed.

This property is read-only.

See Also

Chapter 17, "Feed Classes," DSPARAMETER_INCREMENTAL, page 875

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Incremental Feeds

LINKTYPE_FIRST

Description

Use this property for a paged feed to return a string indicating that link type is first.

This property is read-only.

See Also

Chapter 17, "Feed Classes," getRelativePageLinkForFeed, page 812

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

LINKTYPE_LAST

Description

Use this property for a paged feed to return a string indicating that link type is last.

This property is read-only.

See Also

Chapter 17, "Feed Classes," getRelativePageLinkForFeed, page 812

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

LINKTYPE_NEXT

Description

Use this property for a paged feed to return a string indicating that link type is next.

This property is read-only.

See Also

Chapter 17, "Feed Classes," getRelativePageLinkForFeed, page 812

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

LINKTYPE_PREVIOUS

Description

Use this property for a paged feed to return a string indicating that link type is previous.

This property is read-only.

See Also

Chapter 17, "Feed Classes," getRelativePageLinkForFeed, page 812

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

MIMETYPE_ATOM

Description

Use this property to return a string containing the MIME type for Atom feed format data.

This property is read-only.

MIMETYPE_OPML

Description

Use this property to return a string containing the MIME type for OPML format data.

This property is read-only.

MIMETYPE_XML

Description

Use this property to return a string containing the MIME type as XML for unknown feed format data.

This property is read-only.

OPERATINGMODE_AUTHORIZATION

Description

Use this property to return a number representing the Feed object operating mode as for user authorization only.

Note. Most other operations such as execute, save, and delete are not allowed.

This property is read-only.

OPERATINGMODE_DEFAULT

Description

Use this property to return a number representing the Feed object operating mode as default.

Note. A Feed object in this mode allows all operations such as save, delete, and so on.

This property is read-only.

OPERATINGMODE_DELETION

Description

Use this property to return a number representing the Feed object operating mode as for feed deletion only.

Note. Most other operations such as execute and save are not allowed.

This property is read-only.

OPERATINGMODE_EXECUTION

Description

Use this property to return a number representing the Feed object operating mode as for feed execution only.

Note. Other operations such as save and delete are not allowed.

This property is read-only.

OPERATINGMODE_EXECUTION_NOENTRY

Description

Use this property to return a number representing the Feed object operating mode as for feed execution only to return just an empty feed header (that is, without feed entries).

Note. Otherwise, this mode is the same as the OPERATINGMODE_EXECUTION mode.

This property is read-only.

QUERYPARAMETER_CHILDFEEDID

Description

Use this property to return the ChildFeedID query parameter name as a string.

This property is read-only.

QUERYPARAMETER_DATATYPEID

Description

Use this property to return the PTFP_DATA_TYPE query parameter as a string.

This property is read-only.

QUERYPARAMETER_DEFLOCALNODE

Description

Use this property to return the To query parameter as a string.

This property is read-only.

QUERYPARAMETER_DSSCOUNT

Description

Use this property to return the PTFP_DSS_COUNT query parameter as a string.

This property is read-only.

QUERYPARAMETER_DSSNAME

Description

Use this property to return the PTFP_DSS_NAME query parameter as a string.

This property is read-only.

QUERYPARAMETER_DSSVALUE

Description

Use this property to return the PTFP_DSS_VALUE query parameter as a string.

This property is read-only.

QUERYPARAMETER_FEEDFORMAT

Description

Use this property to return the FeedFormat query parameter as a string.

This property is read-only.

QUERYPARAMETER_FEEDID

Description

Use this property to return the FEED_ID query parameter as a string.

This property is read-only.

QUERYPARAMETER_FEEDLIST

Description

Use this property to return the FEEDLIST query parameter as a string.

This property is read-only.

QUERYPARAMETER_FEEDTYPE

Description

Use this property to return the PTFP_FEED_TYPE query parameter as a string.

This property is read-only.

QUERYPARAMETER_IBTRANSID

Description

Use this property to return the IB_TRANS_ID query parameter as a string.

This property is read-only.

QUERYPARAMETER_IFMODIFIEDSINCE

Description

Use this property to return the If-None-Match query parameter as a string.

This property is read-only.

QUERYPARAMETER_IFNONEMATCH

Description

Use this property to return the PAGE_NUM query parameter as a string.

This property is read-only.

QUERYPARAMETER_KEYWORD

Description

Use this property to return the PTFP_FEED_KEYWORD query parameter as a string.

This property is read-only.

QUERYPARAMETER_LANGUAGE

Description

Use this property to return the languageCd query parameter as a string.

This property is read-only.

QUERYPARAMETER_NODENAME

Description

Use this property to return the NODE_NAME query parameter as a string.

This property is read-only.

QUERYPARAMETER_PAGENUM

Description

Use this property to return this query parameter as a string.

This property is read-only.

QUERYPARAMETER_PORTALNAME

Description

Use this property to return the PORTAL_NAME query parameter as a string.

This property is read-only.

QUERYPARAMETER_PTPPB_SEARCH_MODE

Description

Use this property to return the PTPPB_SEARCH_MODE query parameter as a string.

This property is read-only.

QUERYPARAMETER_PTPPB_SEARCH_TEXT

Description

Use this property to return the SEARCH_TEXT query parameter as a string.

This property is read-only.

RequestInfo

Description

Use this property to set or return a FeedRequest object.

This property is read-write.

SF_MAXMINUTES_ALLMSGS

Description

Use this property to return a number representing *all messages* as the value for the maximum minutes data source parameter for scheduled feeds.

Note. Use the DSPARAMETER_SF_MAXMINUTES property to return the *name* of the maximum minutes data source parameter for scheduled feeds.

This property is read-only.

See Also

Chapter 17, "Feed Classes," DSPARAMETER_SF_MAXMINUTES, page 876

SF_MAXROWOPTION_ALLMSGS

Description

Use this property to return a number representing *all messages* as the value for the maximum row data source parameter for scheduled feeds.

Note. Use the DSPARAMETER_MAXROW property to return the *name* of the maximum row data source parameter for scheduled feeds.

This property is read-only.

See Also

Chapter 17, "Feed Classes," DSPARAMETER_MAXROW, page 876

SF_MAXROWOPTION_LATESTMSG

Description

Use this property to return a number representing *latest message* as the value for the maximum row data source parameter for scheduled feeds.

Note. Use the DSPARAMETER_MAXROW property to return the *name* of the maximum row data source parameter for scheduled feeds.

This property is read-only.

See Also

Chapter 17, "Feed Classes," DSPARAMETER_MAXROW, page 876

SF_PAGINGOPTION_NOPAGING

Description

Use this property to return a number representing *no paging* as the value for the paging data source parameter for scheduled feeds.

Note. Use the DSPARAMETER_SF_PAGING property to return the *name* of the paging data source parameter for scheduled feeds.

This property is read-only.

See Also

Chapter 17, "Feed Classes," DSPARAMETER_SF_PAGING, page 876

SF_PAGINGOPTION_SEGMENTED

Description

Use this property to return a number representing *segmented message* as the value for the paging data source parameter for scheduled feeds.

Note. Use the DSPARAMETER_SF_PAGING property to return the *name* of the paging data source parameter for scheduled feeds.

Important! Using SF_PAGINGOPTION_SEGMENTED returns the oldest page (message segment) as the initial page of the feed. The links to the other pages (next, previous, first, and last) are available in the FeedDoc wherever applicable.

This option is best suited for when the feed output is large and the feed itself is to be consumed by a crawler or a feed reader application that understand these links.

This property is read-only.

See Also

Chapter 17, "Feed Classes," DSPARAMETER_SF_PAGING, page 876

SYSVAR_INVALID

Description

Use this property to return the string that indicates that a system variable is invalid.

This property is read-only.

USAGETYPE_ADMINSPECIFIED

Description

Use this property to return a string indicating that the data source parameter usage type is administrator specified.

This property is read-only.

USAGETYPE_FIXED

Description

Use this property to return a string indicating that the data source parameter usage type is fixed.

This property is read-only.

USAGETYPE_INTERNAL

Description

Use this property to return a string indicating that the data source parameter usage type is internal.

This property is read-only.

USAGETYPE_NOTUSED

Description

Use this property to return a string indicating that the data source parameter usage type is not used.

This property is read-only.

USAGETYPE_SYSVAR

Description

Use this property to return a string indicating that the data source parameter usage type is system variable.

This property is read-only.

USAGETYPE_USERSPECIFIED

Description

Use this property to return a string indicating that the data source parameter usage type is user specified.

This property is read-only.

XMLCHILDELEMENTS_CLOUD

Description

Use this property to return an array of string containing the list of children XML tags used by the FEEDATTRIBUTE_CLOUD feed attribute.

This property is read-only.

See Also

Chapter 17, "Feed Classes," FEEDATTRIBUTE_CLOUD, page 878

XMLCHILDELEMENTS_PERSON

Description

Use this property to return an array of string containing the list of children XML tags used by the FEEDATTRIBUTE_AUTHOR and FEEDATTRIBUTE_CONTRIBUTOR feed attributes.

This property is read-only.

See Also

Chapter 17, "Feed Classes," FEEDATTRIBUTE_AUTHOR, page 878 and Chapter 17, "Feed Classes," FEEDATTRIBUTE_CONTRIBUTOR, page 878

XMLCHILDELEMENTS_TEXTINPUT

Description

Use this property to return an array of string containing the list of children XML tags used by the FEEDATTRIBUTE_TEXTINPUT feed attribute.

This property is read-only.

See Also

Chapter 17, "Feed Classes," FEEDATTRIBUTE_TEXTINPUT, page 881

XMLELEMENT_DAY

Description

Use this property to return as a string the representing the day XML element.

This property is read-only.

XMLELEMENT_DESCRIPTION

Description

Use this property to return as a string the representing the description XML element.

This property is read-only.

XMLELEMENT_DOMAIN

Description

Use this property to return as a string the representing the domain XML element.

This property is read-only.

XMLELEMENT_EMAIL

Description

Use this property to return as a string the representing the email XML element.

This property is read-only.

XMLELEMENT_HOUR

Description

Use this property to return as a string the representing the hour XML element.

This property is read-only.

XMLELEMENT_LINK

Description

Use this property to return as a string the representing the link XML element.

This property is read-only.

XMLELEMENT_NAME

Description

Use this property to return as a string the representing the name XML element.

This property is read-only.

XMLELEMENT_PATH

Description

Use this property to return as a string the representing the path XML element.

This property is read-only.

XMLELEMENT_PORT

Description

Use this property to return as a string the representing the port XML element.

This property is read-only.

XMLELEMENT_PROTOCOL

Description

Use this property to return as a string the representing the protocol XML element.

This property is read-only.

XMLELEMENT_REGISTERPROCEDURE

Description

Use this property to return as a string the representing the register protocol XML element.

This property is read-only.

XMLELEMENT_TITLE

Description

Use this property to return as a string the representing the title XML element.

This property is read-only.

FeedDoc Class

This section provides an overview of the FeedDoc class and discusses:

- FeedDoc class constructor.
- FeedDoc class methods.
- FeedDoc class properties.

The FeedDoc class extends the XmlDoc base class to provide a generic interface class for feed documents. Therefore, for each feed format type, a format-specific feed document class is required to extend the base FeedDoc class. As a base class, the FeedDoc class has abstract methods and properties, which are identified as such in the following sections.

FeedDoc Class Constructor

This section presents the constructor for the FeedDoc class.

FeedDoc

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:XML_FEED:FeedDoc;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:XML_FEED:FeedDoc &this_FeedDoc = &thisFeedFactory.⇒
getFeedDoc("feed_ID", &thisFeedFactory.Utility.FEEDFORMAT_ATOM10, Null);
```

See Also

Chapter 17, "Feed Classes," getFeedDoc, page 860

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

FeedDoc Class Methods

In this section, the FeedDoc class methods are presented in alphabetical order.

addCategory

Syntax

addCategory*(category)*

Description

Use this method to add a category attribute to the feed document as a string.

Note. This is an abstract method.

Parameters

Parameter	*Description*
category	Specifies the category as a string.

Returns

A Boolean value: True if the add was successful, False otherwise.

See Also

Chapter 17, "Feed Classes," deleteCategory, page 906 and Chapter 17, "Feed Classes," Categories, page 910

addEntry

Syntax

`addEntry`*`(entry_ID)`*

Description

Use this method to create a FeedEntry object with the given ID.

Note. This is an abstract method.

Parameters

Parameter	*Description*
entry_ID	Specifies the ID of the feed entry as a string.

Returns

A FeedEntry object if successful, False otherwise.

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:XML_FEED:FeedDoc;
import PTFP_FEED:XML_FEED:FeedEntry;

Local boolean &succeeded;
Local array of string &tempArray;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:XML_FEED:FeedDoc &thisFeedDoc = &thisFeedFactory.Utility.⇒
getFeedDoc("feed_ID", &thisFeedFactory.Utility.FEEDFORMAT_ATOM10, Null);
Local PTFP_FEED:XML_FEED:FeedEntry &thisEntry = &thisFeedDoc.addEntry("entry_ID");

&thisEntry.Title = &contentTitle;
&thisEntry.Description = &contentEntryDesc;
&succeeded = &thisEntry.addCategory(&contentFolderTitle);
&thisEntry.ContentUrl = &contentUrl;
&thisEntry.GUID = &contentUrl;
&thisEntry.Published = &contentCreatedDateTime;
&thisEntry.Updated = &contentLastUpdatedDateTime;
&thisEntry.Author = &thisFeedFactory.Utility.getUserInfo(&contentCreatedByOprid);
&tempArray = &thisFeedFactory.Utility.getUserInfo(&contentLastUpdatedByOprid);
&succeeded = &thisEntry.addContributor(&tempArray [1], &tempArray [2]);
&succeeded = &thisEntry.addEnclosure(&contentAttachmentUrl, ⇒
"application/octet-stream", 123456);
```

See Also

Chapter 17, "Feed Classes," deleteEntry, page 907; Chapter 17, "Feed Classes," getEntry, page 908; Chapter 17, "Feed Classes," resetEntries, page 909 and Chapter 17, "Feed Classes," Entries, page 912

Chapter 17, "Feed Classes," FeedEntry Class, page 917

datetimeToString

Syntax

`datetimeToString`*`(Datetime)`*

Description

Use this method to transform a datetime value to a string in a feed-specific format**—** for example, in the "yyyy-MM-dd HH:mm:ss.SSS" format.

Note. This is an abstract method.

Parameters

Parameter	*Description*
Datetime	Specifies the datetime value to be transformed.

Returns

A string in a feed-specific format or an empty string if the datetime value is empty.

deleteCategory

Syntax

`deleteCategory`*`(category)`*

Description

Use this method to delete a category attribute from a FeedDoc object.

Note. This is an abstract method.

Parameters

Parameter	*Description*
category	Specifies the category as a string.

Returns

A Boolean value: True if the delete was successful, False otherwise.

See Also

Chapter 17, "Feed Classes," addCategory, page 903

deleteEntry

Syntax

deleteEntry(*entry_ID*)

Description

Use this method to delete the feed entry and the FeedEntry object with the given ID.

Parameters

Parameter	*Description*
entry_ID	Specifies the ID of the feed entry as a string.

Returns

A Boolean value: True if the delete was successful, False otherwise.

See Also

Chapter 17, "Feed Classes," addEntry, page 904

Chapter 17, "Feed Classes," FeedEntry Class, page 917 and Chapter 17, "Feed Classes," delete, page 921

equals

Syntax

equals(*&Object*)

Description

Use this method to evaluate the equivalency of an object with the current feed document. Equivalency is established based on the value of two properties: the object's ID and the ObjectType.

Parameters

Parameter	*Description*
&Object	Specifies the object to be compared to the current FeedDoc object.

Returns

A Boolean value: True if the items are equivalent, False otherwise.

See Also

Chapter 17, "Feed Classes," ID, page 914 and Chapter 17, "Feed Classes," ObjectType, page 916

getEntry

Syntax

getEntry*(entry_ID)*

Description

Use this method to return a FeedEntry object with the given ID.

Parameters

Parameter	*Description*
entry_ID	Specifies the ID of the feed entry as a string.

Returns

A FeedEntry object if successful, Null otherwise.

See Also

Chapter 17, "Feed Classes," addEntry, page 904

Chapter 17, "Feed Classes," FeedEntry Class, page 917

resetEntries

Syntax

`resetEntries``()`

Description

Use this method to reset the FeedEntry collection.

Parameters

None.

Returns

An empty FeedEntry collection.

See Also

Chapter 17, "Feed Classes," addEntry, page 904 and Chapter 17, "Feed Classes," Entries, page 912

Chapter 17, "Feed Classes," FeedEntry Class, page 917

stringToDatetime

Syntax

`stringToDatetime``(`*`DatetimeString`*`)`

Description

Use this method to covert a date/time string of a feed-specific format to a datetime value.

Note. This is an abstract method.

Parameters

Parameter	*Description*
DatetimeString	Specifies the string to be converted to a datetime.

Returns

A datetime.

FeedDoc Class Properties

In this section, the FeedDoc class properties are presented in alphabetical order.

AllowMoreEntries

Description

Use this property to return a Boolean value indicating whether more feed entries are allowed for this FeedDoc object.

This property is read-only.

See Also

Chapter 17, "Feed Classes," addEntry, page 904 and Chapter 17, "Feed Classes," MaxEntries, page 915

Chapter 17, "Feed Classes," FeedEntry Class, page 917

ApplicationRelease

Description

Use this property to return the release label from the PSRELEASE table as a string.

This property is read-only.

Categories

Description

Use this property to return the categories for the FeedDoc object as an array of string.

This property is read-only.

Note. This is an abstract property.

See Also

Chapter 17, "Feed Classes," addCategory, page 903

ContentUrl

Description

Use this property to set or return the content URL for the feed document as a string.

This property is read-write.

Note. This is an abstract property.

Example

```
&newDoc.ContentUrl = %This.FeedContentUrl;
```

See Also

Chapter 17, "Feed Classes," FeedContentUrl, page 793

Copyright

Description

Use this property to set or return the copyright for the feed document as a string.

This property is read-write.

Note. This is an abstract property.

Description

Description

Use this property to set or return the description for the feed document as a string.

This property is read-write.

Note. This is an abstract property.

Example

```
&newDoc.Description = %This.Description;
```

See Also

Chapter 17, "Feed Classes," Description, page 791

Entries

Description

Use this property to return a collection of FeedEntry objects.

This property is read-only.

See Also

Chapter 17, "Feed Classes," addEntry, page 904; Chapter 17, "Feed Classes," deleteEntry, page 907 and Chapter 17, "Feed Classes," resetEntries, page 909

Expires

Description

Use this property to set or return the expiration date and time for the feed document as a datetime value.

This property is read-write.

See Also

Chapter 17, "Feed Classes," datetimeToString, page 905 and Chapter 17, "Feed Classes," stringToDatetime, page 909

FeedFormat

Description

Use this property to return the feed format for the feed document as a string.

This property is read-only.

See Also

Chapter 17, "Feed Classes," FeedFormat, page 793

FeedUrl

Description

Use this property to set or return feed URL as a string.

This property is read-write.

Note. This is an abstract property.

Example

```
&newDoc.FeedUrl = &feedUrl;
```

See Also

Chapter 17, "Feed Classes," FeedUrl, page 794

FirstUrl

Description

Use this property to set or return the URL for the first page of a paged feed as a string.

This property is read-write.

See Also

Chapter 17, "Feed Classes," LastUrl, page 914; Chapter 17, "Feed Classes," NextUrl, page 916 and Chapter 17, "Feed Classes," PreviousUrl, page 916

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

Generator

Description

Use this property to set or return the generator of this feed document as a string.

This property is read-write.

Note. This is an abstract property.

See Also

Chapter 17, "Feed Classes," ApplicationRelease, page 910

ID

Description

Use this property to return the ID for this feed document as a string.

This property is read-only.

See Also

Chapter 17, "Feed Classes," FeedDoc, page 903

LastUrl

Description

Use this property to set or return the URL for the last page of a paged feed as a string.

This property is read-write.

See Also

Chapter 17, "Feed Classes," FirstUrl, page 913; Chapter 17, "Feed Classes," NextUrl, page 916 and Chapter 17, "Feed Classes," PreviousUrl, page 916

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

Logo

Description

Use this property to set or return the logo for this feed document as a string.

This property is read-write.

Note. This is an abstract property.

MaxAge

Description

Use this property to set or return the maximum age (in milliseconds) of the feed document as a number.

Note. This property can be defined for scheduled feeds only if the DSPARAMETER_SF_MAXMINUTES data source parameter is set. This property is not enforced by all feed readers.

This property is read-write.

See Also

Chapter 17, "Feed Classes," DSPARAMETER_SF_MAXMINUTES, page 876

MaxEntries

Description

Use this property to set or return the maximum number of feed entries for the feed document as a number.

This property is read-write.

The default value is 0, which is an unlimited number of entries.

See Also

Chapter 17, "Feed Classes," AllowMoreEntries, page 910

Chapter 17, "Feed Classes," FeedEntry Class, page 917

NextUrl

Description

Use this property to set or return the URL for the next page of a paged feed as a string.

This property is read-write.

See Also

Chapter 17, "Feed Classes," FirstUrl, page 913; Chapter 17, "Feed Classes," LastUrl, page 914 and Chapter 17, "Feed Classes," PreviousUrl, page 916

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

ObjectType

Description

Use this property to return the object type for the feed document as a string. For a feed document, this is a constant value: FeedDoc.

This property is read-only.

PreviousUrl

Description

Use this property to set or return the URL for the previous page of a paged feed as a string.

This property is read-write.

See Also

Chapter 17, "Feed Classes," FirstUrl, page 913; Chapter 17, "Feed Classes," LastUrl, page 914 and Chapter 17, "Feed Classes," NextUrl, page 916

PeopleTools 8.51 PeopleBook: Feed Publishing Framework, "Understanding the Feed Publishing Framework," Paged Feeds

RootElement

Description

Use this property to return the root element for this feed document as an XmlNode object.

This property is read-only.

Title

Description

Use this property to set or return the title for the feed document as a string.

This property is read-write.

Note. This is an abstract property.

Example

```
&newDoc.Title = %This.Title;
```

See Also

Chapter 17, "Feed Classes," Title, page 797

Updated

Description

Use this property to set or return the date and time the feed document was last updated as a datetime value.

This property is read-write.

Note. This is an abstract property.

FeedEntry Class

This section provides an overview of the FeedEntry class and discusses:

- FeedEntry class constructor.

- FeedEntry class methods.
- FeedEntry class properties.

The FeedEntry class provides a generic interface class for feed entries. Therefore, for each feed format type, a format-specific feed entry class is required to extend the base FeedEntry class. As a base class, the FeedEntry class has abstract methods and properties, which are identified as such in the following sections.

FeedEntry Class Constructor

This section presents the constructor for the FeedEntry class.

FeedEntry

Example

```
import PTFP_FEED:FeedFactory;
import PTFP_FEED:XML_FEED:FeedDoc;
import PTFP_FEED:XML_FEED:FeedEntry;

Local PTFP_FEED:FeedFactory &thisFeedFactory = create PTFP_FEED:FeedFactory();
Local PTFP_FEED:XML_FEED:FeedDoc &this_FeedDoc = &thisFeedFactory.Utility.⇒
getFeedDoc("feed_ID", &thisFeedFactory.Utility.FEEDFORMAT_ATOM10, Null);
Local PTFP_FEED:XML_FEED:FeedEntry &this_Entry = &this_FeedDoc.⇒
addEntry("entry_ID");
```

See Also

Chapter 17, "Feed Classes," getFeedDoc, page 860

Chapter 17, "Feed Classes," addEntry, page 904

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

FeedEntry Class Methods

In this section, the FeedEntry class methods are presented in alphabetical order.

addCategory

Syntax

`addCategory``(`*`category`*`)`

Description

Use this method to add a category attribute to the feed entry as a string.

Note. This is an abstract method.

Parameters

Parameter	***Description***
category	Specifies the category as a string.

Returns

A Boolean value: True if the add was successful, False otherwise.

See Also

Chapter 17, "Feed Classes," deleteCategory, page 922 and Chapter 17, "Feed Classes," Categories, page 925

addContributor

Syntax

`addContributor``(`*`name,email`*`)`

Description

Use this method to add a contributor attribute to the feed entry.

Note. This is an abstract method.

Parameters

Parameter	*Description*
name	Specifies the name of the contributor as a string.
email	Specifies the email address for the contributor as a string.

Returns

A Boolean value: True if the add was successful, False otherwise.

See Also

Chapter 17, "Feed Classes," deleteContributor, page 922 and Chapter 17, "Feed Classes," Contributors, page 925

addEnclosure

Syntax

`addEnclosure``(`*`URL`*`, `*`type`*`, `*`length`*`)`

Description

Use this method to add an enclosure attribute to the feed entry.

Note. This is an abstract method.

Parameters

Parameter	*Description*
URL	Specifies the URL for the enclosure as a string.
type	Specifies the type of the enclosure as a string.
length	Specifies the length for the enclosure in bytes as a number.

Returns

A Boolean value: True if the add was successful, False otherwise.

See Also

Chapter 17, "Feed Classes," deleteEnclosure, page 923 and Chapter 17, "Feed Classes," Enclosures, page 926

delete

Syntax

delete()

Description

Use this method to delete the feed entry from a feed document.

Parameters

None.

Returns

None.

Example

```
Local boolean &deleted = False;
Local PTFP_FEED:UTILITY:Collection &Entries = %This.Entries;
Local PTFP_FEED:XML_FEED:FeedEntry &thisEntry = %This.getEntry(&pId);

If (&thisEntry <> Null) And
      ( Not &Entries.Immutable) Then
   &deleted = &Entries.deleteById(&pId);
   If &deleted Then
      &thisEntry.delete();
   End-If;
End-If;
```

See Also

Chapter 17, "Feed Classes," deleteEntry, page 907

deleteCategory

Syntax

`deleteCategory`*`(category)`*

Description

Use this method to delete a category attribute from a FeedEntry object.

Note. This is an abstract method.

Parameters

Parameter	***Description***
category	Specifies the category as a string.

Returns

A Boolean value: True if the delete was successful, False otherwise.

See Also

Chapter 17, "Feed Classes," addCategory, page 919

deleteContributor

Syntax

`deleteContributor`*`(name)`*

Description

Use this method to delete a contributor attribute from a feed entry.

Note. This is an abstract method.

Parameters

Parameter	*Description*
name	Specifies the name of the contributor as a string.

Returns

A Boolean value: True if the delete was successful, False otherwise.

See Also

Chapter 17, "Feed Classes," addContributor, page 919

deleteEnclosure

Syntax

deleteEnclosure*(URL)*

Description

Use this method to delete an enclosure attribute from a feed entry.

Note. This is an abstract method.

Parameters

Parameter	*Description*
URL	Specifies the URL for the enclosure as a string.

Returns

A Boolean value: True if the delete was successful, False otherwise.

See Also

Chapter 17, "Feed Classes," addEnclosure, page 920

equals

Syntax

`equals``(&Object)`

Description

Use this method to evaluate the equivalency of an object with the current feed entry. Equivalency is established based on the value of two properties: the object's ID and the ObjectType.

Parameters

Parameter	***Description***
&Object	Specifies the object to be compared to the current FeedEntry object.

Returns

A Boolean value: True if the objects are equivalent, False otherwise.

See Also

Chapter 17, "Feed Classes," ID, page 928 and Chapter 17, "Feed Classes," ObjectType, page 929

FeedEntry Class Properties

In this section, the FeedEntry class properties are presented in alphabetical order.

Author

Description

Use this property to set or return the author attribute for the feed entry as an array of string. The first element of the array is the author's name; the second element is the author's email address.

This property is read-write.

Note. This is an abstract property.

Categories

Description

Use this property to return the categories for the feed entry as an array of string.

This property is read-only.

Note. This is an abstract property.

See Also

Chapter 17, "Feed Classes," addCategory, page 919

Comments

Description

Use this property to set or return a comments attribute for the feed entry as a string.

This property is read-write.

Note. This is an abstract property.

ContentUrl

Description

Use this property to set or return the content URL for the feed entry as a string.

This property is read-write.

Note. This is an abstract property.

Contributors

Description

Use this property to return the contributors for the feed entry as an array of array of string. The first element of the subarray is the contributor's name; the second element is the contributor's email address.

This property is read-only.

Note. This is an abstract property.

See Also

Chapter 17, "Feed Classes," addContributor, page 919

Copyright

Description

Use this property to set or return the copyright for the feed entry as a string.

This property is read-write.

Note. This is an abstract property.

Description

Description

Use this property to set or return the description for the feed entry as a string.

This property is read-write.

Note. This is an abstract property.

Enclosures

Description

Use this property to return the enclosures for the feed entry as an array of array of string. The first element of the subarray is the enclosure's URL; the second element is the enclosure's type; and the third element is the enclosure's length.

This property is read-only.

Note. This is an abstract property.

See Also

Chapter 17, "Feed Classes," addEnclosure, page 920

Expires

Description

Use this property to set or return the expiration date and time for the feed entry as a datetime value.

This property is read-write.

See Also

Chapter 17, "Feed Classes," datetimeToString, page 905 and Chapter 17, "Feed Classes," stringToDatetime, page 909

FeedDoc

Description

Use this property to return the FeedDoc object that is the parent of this feed entry.

This property is read-only.

Example

```
&return_value = %This.FeedDoc.stringToDatetime(&expires_inst.NodeValue);
```

See Also

Chapter 17, "Feed Classes," FeedDoc Class, page 902

FullContent

Description

Use this property to set or return the content for the feed entry as a string.

This property is read-write.

Note. This is an abstract property.

GUID

Description

Use this property to set or return a globally unique ID (GUID) for the feed entry.

This property is read-write.

Note. This is an abstract property. For some feed formats, GUID is set to be the content URL for the feed entry.

See Also

Chapter 17, "Feed Classes," ContentUrl, page 925

ID

Description

Use this property to return the ID for this feed entry as a string.

This property is read-only.

See Also

Chapter 17, "Feed Classes," FeedEntry, page 918

MaxAge

Description

Use this property to set or return the maximum age (in milliseconds) of the feed entry as a number.

Note. This property can be defined for scheduled feeds only if the DSPARAMETER_SF_MAXMINUTES data source parameter is set. This property is not enforced by all feed readers.

This property is read-write.

See Also

Chapter 17, "Feed Classes," DSPARAMETER_SF_MAXMINUTES, page 876

ObjectType

Description

Use this property to return the object type for the feed entry as a string. For a feed entry, this is a constant value: FeedEntry .

This property is read-only.

Published

Description

Use this property to set or return the initial publication date and time for this feed entry as a datetime value.

This property is read-write.

Note. This is an abstract property.

Title

Description

Use this property to set or return the title for the feed entry as a string.

This property is read-write.

Note. This is an abstract property.

Updated

Description

Use this property to set or return the date and time the feed entry was last updated as a datetime value. Initially, the Updated and Published properties have equivalent values. However, if the feed entry is changed, then Updated is different from Published.

This property is read-write.

Note. This is an abstract property.

Additional Feed Examples

This section provides examples of the following: Implementing a real-time feed request handler.

Implementing a Real-Time Feed Request Handler

The following example implements a real-time feed request handler from the IRequestHandler superclass.

Important! For typical real-time feed requests, the default feed service operation (PTFP_GETFEED) and feed request handler should be sufficient. Unless there is specific functionality that is needed for a real-time feed request, use the default feed service operation. For scheduled feeds, the default service operation and hence the request handler cannot be modified.

```
import PS_PT:Integration:IRequestHandler;
import PTFP_FEED:UTILITY:Utility;
import PTFP_FEED:XML_FEED:FeedDoc;
import PTFP_FEED:UTILITY:FeedRequest;
import PTFP_FEED:Interface:IBGetFeedList;
import PTFP_FEED:FeedFactory;
import PTFP_FEED:EXCEPTION:*;


class MyFeedHandler implements PS_PT:Integration:IRequestHandler
   /* --- Properties --- */

   /* --- Methods --- */
   method MyFeedHandler();
   method OnRequest(&pRequestMsg As Message) Returns Message;
   method OnError(&pRequestMsg As Message) Returns string;
end-class;


/** onRequest is defined. Other methods shown as stubs. **/


method MyFeedHandler

   /* method stub */

end-method;


method OnError
   /+ &pRequestMsg as Message +/
   /+ Returns String +/
   /+ Extends/implements PS_PT:Integration:IRequestHandler.OnError +/
   Local string &errstring;
   /* method stub */
   Return &errstring;
end-method;


/**
  * onRequest
  *
  * @param pRequestMsg Request Message.
  *
  * @exception FeedException thrown if switch user failed.
  *
  * @return Message Feed Document.
  *
  */
method OnRequest
   /+ &pRequestMsg as Message +/
   /+ Returns Message +/
   /+ Extends/implements PS_PT:Integration:IRequestHandler.OnRequest +/

   Local boolean &succeeded;
   Local integer &i;
   Local string &temp, &name, &value;
   Local string &errorText;
   Local string &ibTransID, &pageNum, &HTTP_IfNoneMatch, &HTTP_IfModifiedSince,⇒
 &DefaultLocalNode;

   Local Message &responseMsg;
   Local XmlDoc &xmlDoc;
```



```
Local PTFP_FEED:FeedFactory &feedFactory_inst;
Local PTFP_FEED:UTILITY:Utility &utility = &feedFactory_inst.Utility;
Local PTFP_FEED:XML_FEED:FeedDoc &feedDoc;
Local PTFP_FEED:UTILITY:FeedRequest &request;
Local PTFP_FEED:Interface:IBGetFeedList &feedListObj;


/* Ccreate the Search Request object */
&request = create PTFP_FEED:UTILITY:FeedRequest("FeedRequest");


/* Get the search criteria */
For &i = 1 To &pRequestMsg.IBInfo.IBConnectorInfo.GetNumberOfQueryStringArgs()

   &name = &pRequestMsg.IBInfo.IBConnectorInfo.GetQueryStringArgName(&i);
   &value = &pRequestMsg.IBInfo.IBConnectorInfo.GetQueryStringArgValue(&i);
   &succeeded = &request.addParameter(&name, &value);


   Evaluate Upper(&name)
   When Upper(&utility.QUERYPARAMETER_FEEDLIST)
      /* Feed List Request, forward the call */
      &feedListObj = create PTFP_FEED:Interface:IBGetFeedList();
      Return &feedListObj.OnRequest(&pRequestMsg);

   When Upper(&utility.QUERYPARAMETER_PORTALNAME)
      &request.PortalName = &value;
      Break;

   When Upper(&utility.QUERYPARAMETER_NODENAME)
      &request.NodeName = &value;
      Break;

   When Upper(&utility.QUERYPARAMETER_DATATYPEID)
      &request.DataTypeID = &value;
      Break;

   When Upper(&utility.QUERYPARAMETER_FEEDID)
      &request.FeedID = &value;
      Break;

   When Upper(&utility.QUERYPARAMETER_LANGUAGE)
      &request.LanguageCode = &value;
      Break;

   When Upper(&utility.QUERYPARAMETER_IBTRANSID)
      &ibTransID = &value;
      Break;

   When Upper(&utility.QUERYPARAMETER_PAGENUM)
      &pageNum = &value;
      Break;

   When Upper(&utility.QUERYPARAMETER_DEFLOCALNODE)
      &DefaultLocalNode = &value;
      Break;

      /* HTTP Conditonal Headers */
   When Upper(&utility.QUERYPARAMETER_IFNONEMATCH)
      &HTTP_IfNoneMatch = &value;
      Break;

   When Upper(&utility.QUERYPARAMETER_IFMODIFIEDSINCE)
      &HTTP_IfModifiedSince = &value;
```



```
            Break;

         End-Evaluate;
      End-For;


      /* Get the current user PS_TOKEN */
      &request.PS_TOKEN = GenToken();


      /* Get the FeedDoc */
      &errorText = "";

      try

         &feedDoc = &feedFactory_inst.getFeedDoc(&request);

      catch PTFP_FEED:EXCEPTION:NotFoundException &ex1
         &errorText = MsgGetExplainText(219, 3112, "(Message not found) Not Found");

      catch PTFP_FEED:EXCEPTION:PrivilegeException &ex2
         &errorText = MsgGetExplainText(219, 3113, "(Message not found) ⇒
   Not Authorized");

      catch PTFP_FEED:EXCEPTION:FeedException &ex3
         &errorText = &utility.getExceptionText(&ex3);

      end-try;


      /* Create the response message */
      &responseMsg = CreateMessage(Operation.PTFP_GETFEED, %IntBroker_Response);

      If None(&errorText) Then
         &responseMsg = &utility.setMessageHeadersAndMimeType(&responseMsg, &feedDoc,⇒
    &request);
      Else
         &temp = "<?xml version='1.0' encoding='UTF-8'?><ErrorMessage>" | &errorText⇒
    | "</ErrorMessage>";
         &xmlDoc = CreateXmlDoc(&temp);
         &responseMsg.SetXmlDoc(&xmlDoc);
         &responseMsg.SegmentContentType = &utility.MIMETYPE_XML;
      End-If;

      Return &responseMsg;

   end-method;
```

Chapter 18

Field Class

This chapter provides an overview of Field class and discusses the following topics:

- Shortcut considerations
- Data Type for a field object
- Scope of a field object
- Field class built-in function
- Field class methods
- Field class properties

Understanding the Field Class

A *field* object, instantiated from the field class, is a single instance of data within a record object, and is based on a field definition.

Accessing or changing the value of a field using the Value property is a common action.

SetDefault is a frequently used method. Name, Enabled, and Type are several commonly used field properties. If you are working with message objects, EditError is also a commonly used field property.

The field class is one of the data buffer access classes.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Considerations Using User Interface Properties

All of the field properties related to the user interface, such as the Label property, only return the value previously set in PeopleCode (false, or blank if nothing was set). They do not return the values set for the field in Application Designer.

Shortcut Considerations

An expression of the form

`FIELD.`*`fieldname.property`*

or

`FIELD.`*`fieldname.method`*`(. . .)`

is converted to an object expression by using `GetField(`**`FIELD.`***`fieldname)`*. For example, the next two lines of code are identical:

```
FIELD.CHECKLIST_DT.Enabled = False;
```

```
GetField(FIELD.CHECKLIST_DT).Enabled = False;
```

An expression of the form

`recname.fieldname.property`

or

`recname.fieldname.method``(.  .  .)`

is converted to an object expression by using `GetField`*`(recname.fieldname)`*. For example, the next two lines of code are identical:

```
EMPL_CHECKLINST.CHECKLIST_DT.Enabled = False;
```

```
GetField(EMPL_CHECKLIST.CHECKLIST_DT).Enabled = False;
```

Data Type for a Field Object

Field objects are declared as type Field. For example,

```
Local Field &MyField;
```

Scope of a Field Object

A field can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

Field Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetField

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetPageField

Field Class Methods

In this section, we discuss the Field class methods in alphabetical order.

AddDropDownItem

Syntax

`AddDropDownItem``(`*`CodeString`*`,`*`DescriptionString`*`)`

Description

The AddDropDownItem method adds an item to the dropdown list in the control for the field. The first time this method is called, it overrides the prompt table or translate table used to populate the list. Those items no longer appear in the list. Only the items added using this method display.

Subsequent calls to this method adds additional items to the dropdown list. The items added with the first call to the method also display.

If there is an existing value and the dropdown list is changed with these functions, the selection shows as (Invalid value) unless the new list contains an entry with the same code as the existing value.

Considerations Using AddDropDownItem

If the data for the dropdown is language sensitive, the values for the dropdown should come from the message catalog or from a database field that has a related language record, and should not be hard-coded.

A good place for your PeopleCode program to populate a dropdown list is in the RowInit event. This event executes before the page is shown for the first time, so it prevents unnecessary SQL.

Parameters

Parameter	***Description***
CodeString	Specify the value used to set the field value if this item is selected. Codes longer than the size of the field are truncated.
DescriptionString	Specify the value the end-user sees in the dropdown list.

Returns

None.

Example

Using a hardcoded list is not appropriate for this function because translations do not work. The data must come from the Translate Table (or other record) directly so that the data is translated correctly.

```
Local Rowset &Xlat;

&FLD = GetRecord(Record.JOB).GetField(Field.ACTION);
&FLD.ClearDropDownList();

Evaluate %Component
When Component.JOB_DATA_CONCUR
   &Xlat = CreateRowset(Record.PSXLATITEM);
   &Xlat.Fill("WHERE FILL.FIELDNAME = 'ACTION' AND Fill.FIELDVALUE in ⇒
('ADL','HIR') and EFFDT = (select max(EFFDT) from PSXLATITEM B where B.FIELDNAME =⇒
 'ACTION' and B.FIELDVALUE in ('ADL','HIR') and EFFDT <= JOB.EFFDT)");

   &Xlat_cnt = &Xlat.ActiveRowCount;
   For &I = 1 To &Xlat_cnt
      &CodeIn = &Xlat.GetRow(&I).GetRecord(1).FIELDVALUE.Value;
      &DescIn = &Xlat.GetRow(&I).GetRecord(1).XLATLONGNAME.Value;

      &FLD.AddDropDownItem(&CodeIn, &DescIn);
   End-For;

   Break;
When-Other
End-Evaluate;
```

See Also

Chapter 18, "Field Class," ClearDropDownList, page 938

ClearDropDownList

Syntax

ClearDropDownList()

Description

The ClearDropDownList method clears all items added to the dropdown list using the AddDropDownItem method. In addition, this method causes the prompt table or translate table values defined for the list to come back into effect again (unless they're subsequently overridden again with AddDropDownItem.)

Parameters

None.

Returns

None.

Example

```
&FLD = GetRecord(Record.ABSENCE_HIST).GetField(Field.ABSENCE_TYPE);
&FLD.AddDropDownItem("CNF", "Conference");
&FLD.AddDropDownItem("VAC", "Vacation");
&FLD.AddDropDownItem("SCK", "Sick");
. . . .
&FLD.ClearDropDownList();
```

See Also

Chapter 18, "Field Class," AddDropDownItem, page 937

DecryptPETKey

Syntax

DecryptPETKey()

Description

Use this method to apply PeopleSoft Encryption Technology (PET) to decrypt an PET-encrypted key.

Returns

None.

Example

```
If %Component = Component.CRYPT_KEYSET Then
   &rec = GetRecord(Record.PSCRYPTKEYSET);
   DERIVED_CRYPT.CRYPT_KEY = &rec.CRYPT_KEY.DecryptPETKey();
End-If;
```

See Also

Chapter 18, "Field Class," EncryptPETKey, page 940

PeopleTools 8.51 PeopleBook: Security Administration, "Securing Data with PeopleSoft Encryption Technology"

EncryptPETKey

Syntax

EncryptPETKey()

Description

Use this method to apply PeopleSoft Encryption Technology (PET) to PET-encrypt a key.

Returns

None.

Example

```
If %Component = Component.CRYPT_KEYSET Then;
   &crypt_key = GetField(Field.CRYPT_KEY);
   PSCRYPTKEYSET.CRYPT_KEY = &crypt_key.EncryptPETKey();
End-If;
```

See Also

Chapter 18, "Field Class," DecryptPETKey, page 939

PeopleTools 8.51 PeopleBook: Security Administration, "Securing Data with PeopleSoft Encryption Technology"

GetAuxFlag

Syntax

GetAuxFlag*(FlagNumber)*

Description

Use the GetAuxFlag method to determine whether the Auxiliary Flag Mask specified by *FlagNumber* has been set for a field.

Currently, only one flag comes preset from PeopleSoft: a 1 indicates a ChartField.

Parameters

Parameter	*Description*
FlagNumber	Specify the flag number. 1 is a ChartField. Valid values are between 1 and 32.

Returns

A Boolean value: True, the flag is set, False if the flag hasn't been set.

Example

```
&field = GetField(Field.RC_TEST_PB);
&ret = &field.GetAuxFlag(1);
If (&ret = True) Then
   MessageBox(0, "Metadata Fn Status", 0, 0, "Aux Flag 1 is SET - ChartField");
Else
   MessageBox(0, "Metadata Fn Status", 0, 0, "Aux Flag 1 is NOT SET - Not a Chart⇒
Field");
End-If;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SetDBFieldAuxFlag

GetLongLabel

Syntax

GetLongLabel*(LabelID)*

Description

The GetLongLabel method returns the long name for a field given a label ID. If the given label ID isn't found, a null string is returned. *LabelID* takes a string value.

Note. If a button is defined as an HTML button or hyperlink, and if it has an associated record field, the label associated with it is only the text of the button. The mouse-over text can only be changed by the lable if the button is defined as a button with an Image label.

Returns

A text string containing the long name of the field for the specified label ID.

Example

The following code sets the label for a field to one of two different texts, based on the page.

```
Local Field &FIELD;

&FIELD = GetField(RECORD.MYFIELD);
   If %Page = PAGE.PAGE1 Then
      &FIELD.Label = &FIELD.GetLongLabel("LABEL1");
   Else If %Page = PAGE.PAGE2 Then
      &FIELD.Label = &FIELD.GetLongLabel("LABEL2");
End-If;
```

If the Label ID is the same as the name of the field, you could use the following:

```
&LABELID = &FIELD.Name;
&FIELD.Label = &FIELD.GetLongLabel(&LABELID);
```

See Also

Chapter 18, "Field Class," GetShortLabel, page 943; Chapter 18, "Field Class," Label, page 960 and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," SetLabel

GetRelated

Syntax

GetRelated*(recname.fieldname)*

Description

The GetRelated method returns a field object for a related field *recname.fieldname* that has the field executing the method as its control field.

This method is similar to the GetRelField built-in function, however, the built-in works only in the current context, while this method can be applied to a field from any position in the buffer structure.

Using GetRelated With a Control Field

PeopleCode events on the Control Field can be triggered by the Related Edit field. When this happens, there can be different behavior than with other types of fields:

- If the events are called from FieldEdit of the Control Field, and that FieldEdit is triggered by a change in the Related Edit field, the functions return the previous value.
- If the events are called from FieldChange of the Control Field, and that FieldChange is triggered by a change in the Related Edit field, the functions return the value entered into the Related Edit. This may be a partial value, that will subsequently be expanded to a complete value when the processing is complete.

Parameters

Parameter	*Description*
recname.fieldname	Specifies a field in the same row as the current field object, that has the field executing the method as its control field.

Returns

The field object for the field with the specified name that is related to the field executing the method.

Example

In the following example, the field object is instantiated, then the related display field object. The Value property for the related display is changed, it is disabled, and another variable is assigned its value.

```
Local Field &FIELD, &REL_FIELD;

&FIELD = GetField(OPC_9A2FIELDS.COMPANY);
/* control field object  */
&REL_FIELD = &FIELD.GetRelated(COMPANY_TBL.DESCR);
/* related display object */
&REL_FIELD.Value = "Change";
&REL_FIELD.Enabled = False;
&TMP = &REL_FIELD.Name;
&REL_FIELD.Value = &TMP;
```

If you were not going to use the &FIELD variable later, the first two lines of code in the previous example could be combined:

```
&REL_FIELD = GetField(OPC_9A2FIELDS.COMPANY).GetRelated(COMPANY_TBL.DESCR);
```

Suppose you had two control fields, EMPLID and MANAGER_ID, and both use the NAME field on the PERSONAL_DATA table as their related display. If you need to access both related display fields, you could do the following:

```
&NAME_EMPLID = GetField(PERSONAL_DATA.EMPLID).GetRelated(PERSONAL_DATA.NAME);

&NAME_MANAGER = GetField(PERSONAL_DATA.MANAGER_ID).GetRelated(PERSONAL_DATA.NAME);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetField

GetShortLabel

Syntax

GetShortLabel*(LabelID)*

Description

The GetShortLabel method returns the short name for a field given a label ID. If the given label ID isn't found, a null string is returned. *LabelID* takes a string value.

Note. If a button is defined as an HTML button or hyperlink, and if it has an associated record field, the label associated with it is only the text of the button. The mouse-over text can only be changed by the lable if the button is defined as a button with an Image label.

Returns

A text string containing the short name of the field for the specified label ID.

Example

The following code sets the label for a field to one of two different texts, based on the page.

```
Local Field &FIELD;

&FIELD = GetField(RECORD.MYFIELD);
   If %Page = PAGE.PAGE1 Then
      &FIELD.Label = &FIELD.GetShortLabel("LABEL1");
   Else If %Page = PAGE.PAGE2 Then
      &FIELD.Label = &FIELD.GetShortLabel("LABEL2");
End-If;
```

If the Label ID is the same as the name of the field, you could use the following:

```
&LABELID = &FIELD.Name;
&FIELD.Label = &FIELD.GetShortLabel(&LABELID);
```

See Also

Chapter 18, "Field Class," GetLongLabel, page 941; Chapter 18, "Field Class," Label, page 960 and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference,* "PeopleCode Built-in Functions," SetLabel

SearchClear

Syntax

`SearchClear()`

Description

The SearchClear method clears the field values if the field is a search key.

Considerations Using SearchClear and SetDefault

The SetDefault method causes a field to be set to its default value (if there is one) immediately after SearchInit PeopleCode finishes. SetDefault overrides SearchClear. If you call SearchClear for a record, then use SetDefault for a field, the field is set to its default value and the search key values for the rest of the record are cleared.

Parameters

None.

Returns

None.

Example

```
Local Field &FIELD;

&FIELD = GetField(PERSONAL_DATA.EMPLID);
&FIELD.SearchClear();
```

See Also

Chapter 18, "Field Class," SetDefault, page 946 and Chapter 34, "Record Class," SearchClear, page 2095

SetCursorPos

Syntax

SetCursorPos(PAGE.*pagename* **| %Page**)

Description

The SetCursorPos method enables you to set the focus to the page field corresponding to the field object (on the specified page). The current page may be specified as %Page.

Restrictions on Use with a Component Interface

This function is ignored (has no effect) when used by a PeopleCode program that's been called by a Component Interface.

Parameters

Parameter	*Description*
Pagename	The name of the page, preceded by the keyword **Page.** The *pagename* page must be in the current component. You can also pass the %page system variable in this parameter (without the **Page** reserved word).

Returns

None.

Example

The following pseudo-code enables you to set the focus to a related field:

```
GetField(ControlRec.ControlField).GetRelated(RelatedRec.RelatedField).SetCursorPos⇒
(PAGE.pagename);
```

See Also

Chapter 18, "Field Class," SetDefault, page 946 and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference,* "PeopleCode Built-in Functions," SetCursorPos

SetDefault

Syntax

```
SetDefault()
```

Description

SetDefault sets the value of the field to a null value, or to a default, depending on the type of field.

- If this method is used against data from the component buffers, the next time default processing occurs, it is set to its default value: either a default specified in its record field definition or one set programmatically by PeopleCode located in a FieldDefault event. If neither of these defaults exist, the Component Processor leaves the field blank.

- If this method is used with a field that isn't part of the data buffer (for example, a field in a record object instantiated with CreateRecord) the field is automatically set to its default value if one is set for the *field,* not for the record field. Any FieldDefault PeopleCode will not be run on these types of fields. If you want to set the default values for all the fields in a record, use the SetDefault record class method.

Blank numbers correspond to zero on the database. Blank characters correspond to a space on the database. Blank dates and long characters correspond to NULL on the database. SetDefault gives each field data type its proper value.

Parameters

None.

Returns

None.

Example

```
&CHARACTER.SetDefault();
```

See Also

Chapter 34, "Record Class," SetDefault, page 2099 and *PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide*, "PeopleCode and the Component Processor," Default Processing

Field Class Properties

In this section, we discuss the Field class properties. The properties are discussed in alphabetical order.

DataAreaCollapsed

Description

This property specifies whether the default initial view of the data area of a group box is collapsed or expanded.

Note. You must set the Collapsible Data Area on the properties for the group box in Application Designer for this property to have any effect.

This property changes to reflect the current state of the data area, according to whether the user has collapsed or expanded it. Changing the value collapses or expands the data area, but it does *not* prevent the user from collapsing (or expanding) it themselves.

Note. Because the user can change the value of this property, whatever value is set in PeopleCode isn't guaranteed to be still set the next time it is checked, because the user may have collapsed or expanded the data area in the meantime.

This property overwrites the value of the Default Initial View to Expanded field set in Application Designer. For example, if Default Initial View to Expanded is selected in Application Designer, then the value for the DataAreaCollapsed property is set to True, the control initially displays collapsed.

This property takes a Boolean value: True, initially display the data area collapsed, False, initially display the data area expanded.

This property is read-write.

Note. If you want to collapse an entire level-based control, such as a scroll area or a grid, use the DataAreaCollapsed Rowset method.

See Also

Chapter 36, "Rowset Class," DataAreaCollapsed, page 2162

DecimalPosition

Description

Use this property to specify the number of digits after the decimal point to be displayed for a field defined as number or signed number.

This property *overwrites* the Decimal Positions value defined for a field in Application Designer.

Setting the DecimalPosition property to a smaller number than the record field's decimal position causes the displayed data to be truncated. For example, suppose a numeric field in the database has a value 0.005. Setting DecimalPosition to 2 displays 0.00 on page.

This property takes an integer value. The default value is -1, which indicates that the property is not set.

This property is read-write.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Field Definitions," Specifying Number Field Attributes

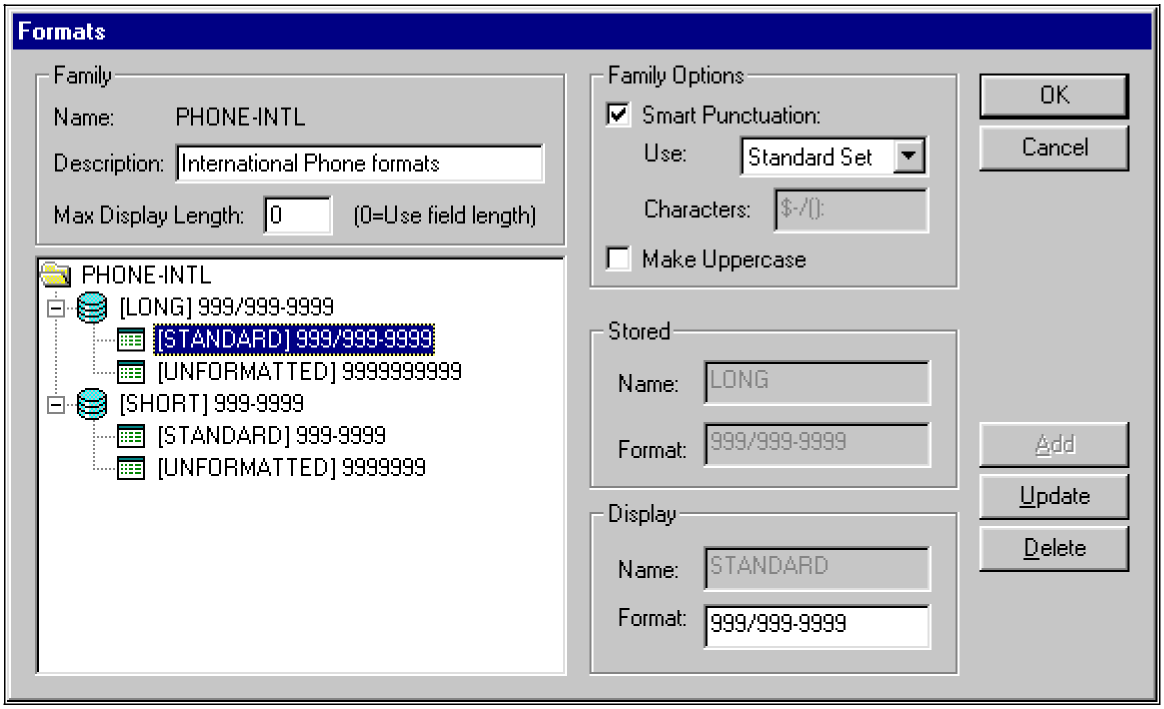
DisplayFormat

Description

Use this property to specify a custom format to use for the field.

This property is read-write.

The custom format for a field is specified in the field definition. This property enables you to switch between display formats that are defined as part of a custom format. For example, suppose your field used the PHONE-INTL custom format:



PHONE-INTL format

Both the LONG and the SHORT stored formats have two display formats: STANDARD and UNFORMATTED.

Using this property, you could select either of the display formats. For example:

```
If %Component = PAGE.INTERNATIONAL
   &CHAR.DisplayFormat = "STANDARD";
Else
   &CHAR.DisplayFormat = "UNFORMATTED";
End-If;
```

Note. You *cannot* change the Stored format, but you can find its value using the StoredFormat property.

Example

```
&CHARACTER.DisplayFormat = "STANDARD";

/* this assumes that &CHARACTER is a custom formatted field,
and a display format option is STANDARD */
```

See Also

Chapter 18, "Field Class," StoredFormat, page 970

DisplayOnly

Description

This property is set to true if the field property DisplayOnly in the Application Designer is set. May be set to false to change how the field displays.

Note. This property *overwrites* whatever value is set in Application Designer.

This property is read-write.

DisplayZero

Description

Use this property to specify if, in a numeric field, zeros or blank are displayed on a page.

This property *overwrites* the Display Zero value on the page field properties dialog box in Application Designer.

This property takes a Boolean value: true, display zeros, false, display blank. The default values is false.

The value of the DisplayZero property can be changed by the SmartZero property.

If SmartZero is set to true, and your application sets DisplayZero to false in a PeopleCode program, the application server changes the value of DisplayZero to true to display zeros, not blanks. When this occurs, the DisplayZeroChanged property is set to true.

This property is read-write.

See Also

Chapter 18, "Field Class," DisplayZeroChanged, page 950 and Chapter 18, "Field Class," SmartZero, page 969

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Field Definitions," Specifying Attributes for a New Field Definition

DisplayZeroChanged

Description

Use this property to determine if the original DisplayZero property has been changed from its original setting.

The DisplayZeroChanged property is only relevant when the SmartZero property has been set to true. When SmartZero equals true, the application server adjusts the DisplayZero property to display a blank or zero based on the user's input. You can then check the DisplayZeroChanged property to find out if the application server has changed the DisplayZero property.

This property takes a Boolean value: true, the application server has changed the field's DisplayZero property, false otherwise.

For example, suppose you have a PeopleCode program in the RowInit event that sets the DisplayZero property for a field to True, and also sets the SmartZero property to True. When a user deletes the numeric zero in that field on the page, and the page is submitted to the application server, the application server sets the field's DisplayZero property to False so that the new page displays a blank, not a numeric zero. The DisplayZeroChanged property is set to true in this example.

This property is read-only.

See Also

Chapter 18, "Field Class," DisplayZero, page 950 and Chapter 18, "Field Class," SmartZero, page 969

EditError

Description

This property is True if an error for this field has been found after executing the ExecuteEdits method with either a message object or a record object. This property can be used with the MessageSetNumber and MessageNumber properties to find the error message set number and error message number.

The EditError property returns True if a Null value was encountered for this field.

This property is read-write. After you have fixed the errors, you must set this property to False before running ExecuteEdits again.

Example

The following is an example showing how EditError, along with the ExecuteEdits method could be used:

```
&REC.ExecuteEdits();
If &REC.IsEditError Then
   For &I = 1 to &REC.FieldCount
      If &REC.GetField(&I).EditError Then
         LOG_ERROR();  /* application specific pgm */
      End-If;
   End-For;
End-If;
```

See Also

Chapter 34, "Record Class," ExecuteEdits, page 2086; Chapter 25, "Message Classes," ExecuteEdits, page 1268; Chapter 18, "Field Class," MessageNumber, page 962 and Chapter 18, "Field Class," MessageSetNumber, page 963

Enabled

Description

This property is True if this field is enabled. May be set False to disable any control that displays this field.

This property is read-write.

Example

```
&CHARACTER.Enabled = True;

&CHARACTER.Enabled = False;
```

FieldLength

Description

This property returns the length of the field as a number.

This property is read-only.

Example

```
&MyRec = CreateRecord(Record.BKREC1);
&MyField = &MyRec.GetField(Field.BKNAME);
&length = &MyField.FieldLength;
MessageBox(0, "Field Length", 0, 0, "The field BKREC1.BKNAME is length " |⇒
 &length);
```

FormatLength

Description

This property returns the length of the format for the field as a number.

This property is read-only.

Example

```
&field = GetField(Field.BKTEST);
&formatlength = &field.FormatLength;
&length = &field.FieldLength;
&mymsg = "Field.BKTEST length is " | &length | " and format length is " |⇒
 &formatlength;
MessageBox(0, "MetaData Fn Status", 0, 0, &mymsg);
```

FormattedValue

Description

This property returns the value of a field as a string, formatted exactly as it would be displayed on an edit field on a page. This is useful when you're using a prompt field for the label of another field.

Because a record field can be bound to more than one page field, FormattedValue takes the first associated page field, looking first on the current page, then through all the pages in the component. This property returns a null string ("") if used with an Image or ContentReference field. It also returns a null string if no associated page field is found.

This property is read-write.

Example

This property could be used to set up a hyperlink labeled by data generated on a page. To do this, do the following:

1. Define a work field.
2. Associate the work field with the hyperlink page field.
3. Define a hidden page field with the formatting that you want.
4. Associate the hidden page field with the data field.
5. Write PeopleCode (probably for RowInit) to set the label on the work field to be the FormattedValue of the data field.

For example, suppose you wanted a hyperlink labeled by the QE_POSITION_ID field on the QE_PLYR_POSITN record. This field is a prompt table field, edited by table QE_POSITION.

You want to display the DESCRSHORT field from the prompt table, instead of the QE_POSITION_ID code value. Create a work field, POS_LINK on the WORK_REC record, and create a hyperlink page field associated with the DESCRSHORT field. For the label, create an invisible drop-down list page field, associated with QE_PLYR_POSITN.QE_POSITION_ID. Set its Prompt Table Field to DESCRSHORT. Then put a PeopleCode program on the component RowInit for either the QE_PLYR_POSITN record or the WORK_REC.

```
WORKREC.POS_LINK.Label = QE_PLYR_POSITN.QE_POSITION_ID.FormattedValue;
```


HoverText

Description

Use this property to override the hover text for any push buttons or hyperlinks associated with the field. The maximum length of hover text is 100 characters. If the hover text is identical to the push button or hyperlink label, the hover text will not be shown.

Important! Even if the hover text has been set in Application Designer, this property returns a blank string if you use the property before you've explicitly set it in PeopleCode. This property overrides the existing value only. When a user navigates away from the component, and then back to the page again, the original value is used until it's changed again through PeopleCode.

This property is read-write.

Example

```
QE_ABSENCE_HIST.QE_ABSENCE_TYPE.HoverText = MsgGetText(95, 5037, "Personalize⇒
 Layout");
```

See Also

Chapter 18, "Field Class," Label, page 960

IsAltKey

Description

This property is True if this field references a field definition that is defined as an Alternate Search key in the associated record definition.

This property is read-only.

IsAuditFieldAdd

Description

This property is True if this field references a field definition that is defined as an audit field in the associated record definition, and the field is audited whenever new data is added.

This property is read-only.

IsAuditFieldChg

Description

This property is True if this field references a field definition that is defined as an audit field in the associated record definition, and the field is audited whenever data is changed.

This property is read-only.

IsAuditFieldDel

Description

This property is True if this field references a field definition that is defined as an audit field in the associated record definition, and the field is audited whenever data is deleted.

This property is read-only.

IsAutoUpdate

Description

This property is True if this field references a field definition that is defined as an Auto-Update field in the associated record definition.

This property is read-only.

IsChanged

Description

This property returns True if the value for the field has been changed.

Note. Use this property with the primary database record only. It will not return valid results if used with a work record.

This property is read-only.

Example

```
If &CHARACTER.IsChanged Then

     Warning ("The character field has been changed");
```



```
End-If;
```

IsDateRangeEdit

Description

This property is True if this field references a field definition that is defined as requiring a Reasonable Date in the associated record definition.

This property is read-only.

IsDescKey

Description

This property is True if this field references a field definition that is defined as a descending key in the associated record definition.

This property is read-only.

IsDuplKey

Description

This property is True if this field references a field definition that is defined as a duplicate key in the associated record definition.

This property is read-only.

IsEditTable

Description

This property is True if this field references a field definition that has a table edit type of a Prompt Table with Edit.

This property is read-only.

IsEditXlat

Description

This property is True if this field references a field definition that has a table edit type of a Translate Table Edit.

This property is read-only.

IsFromSearchField

Description

This property is True if this field references a field definition that is defined as a From Search Field in the associated record definition.

This property is read-only.

IsInBuf

Description

This property returns True if the data of the field is present in the data buffers. For example, all of the fields of a derived record are not present in the data buffer.

This property is read-only.

Example

The following example iterates over all the fields in a record. The code verifies that the data for the field is accessible before it tries to assign the value to a variable.

```
For &I = 1 to &REC.FieldCount
   &FIELD = &REC.GetField(&I);
   If &FIELD.IsInBuf Then
      &VALUE = &FIELD.Value;
      /* do other processing */
   End-If;
End-For;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Referencing Data in the Component Buffer," Record Fields and the Component Buffer

IsKey

Description

This property is True if this field references a field definition that is defined as a primary key of the associated record definition.

This property is read-only.

Example

```
If &CHARACTER.IsKey Then

     Warning ("The field " | &Character.Name | " is a key");

End-If;
```

IsListItem

Description

This property is True if this field references a field definition that is defined as a List Box item of the associated record definition.

This property is read-only.

IsNotUsed

Description

This property is True if this field has been set as NotUsed with the SetDBFieldNotUsed function.

This property is read-only.

Example

```
&field = GetField(Field.RC_TEST_PB);
&ret = &field.IsNotUsed;
If (&ret = True) Then
   MessageBox(0, "MetaData Fn Status", 0, 0, "Field is NOTUSED");
Else
   MessageBox(0, "MetaData Fn Status", 0, 0, "Field is used.");
End-If;
```


IsRequired

Description

This property is True if this field references a field definition that is defined as Required in the associated record definition.

This property is read-only.

IsRichTextEnabled

Description

This property is True if this field is a long edit box with the rich text editor enabled.

This property is read-only.

IsSearchItem

Description

This property is True if this field references a field definition that is defined as a Search key in the associated record definition.

This property is read-only.

IsSystem

Description

This property is True if this field references a field definition that is defined as System Maintained field in the associated record definition.

This property is read-only.

IsThroughSearchField

Description

This property is True if this field references a field definition that is defined as a Through Search Field in the associated record definition.

This property is read-only.

IsUseDefaultLabel

Description

This property is True if this field references a field definition that has its label defined as Use Default Label in the associated record definition.

This property is read-only.

IsYesNo

Description

This property is True if this field references a field definition that has a table edit type of a Yes/No Table Edit.

This property is read-only.

Label

Description

Use this property to override the label text for the field when it's displayed on a page.

Important! Even if the label text has been set in Application Designer, this property returns a blank string if you use the property before you've explicitly set it in PeopleCode. This property overrides the existing value only. When a user navigates away from the component, and then back to the page again, the original value is used until it's changed again through PeopleCode.

To set the label in PeopleCode, use this property or the SetLabel built-in function. To return the text of a label before you set it, use the GetShortLabel or GetLongLabel methods.

Note. You can't use this property to set labels longer than 100 characters. If you try to set a label of more than 100 characters, the label is truncated to 100 characters.

The "tool tip," or mouse over text, that appears with a hyperlink at runtime comes from the RFT long label assigned to the record field. However, the RFT long label displays only if it is different from the assigned display value of the hyperlink and it is not null. If the link is an image button, the tool tip is derived from the label text if there is any. Otherwise, the RFT long label is used.

This property is read-write.

Example

The following code sample changes all the field labels for a record to the short label. It assumes that there is a label with the same name as the name of the field for all fields in the record.

```
Local Field &FLD;
Local Record &REC;

If CHECK_FIELD Then
   &REC = GetRecord();
   For &I = 1 to &REC.FieldCount
      &FLD = &REC.GetField(&I);
      &LABELID = &FLD.Name;
      &FLD.Label = &FLD.GetShortLabel(&LABELID);
   End-For;
End-If;
```

See Also

Chapter 18, "Field Class," GetShortLabel, page 943 and Chapter 18, "Field Class," GetLongLabel, page 941

LabelImage

Description

Use this property to override the image for a button. This property is only valid for button controls. This property can be set even when the button is disabled or display-only.

Important! Even if the image has been set in Application Designer, this property returns a blank string if you use the property before you've explicitly set it in PeopleCode. This property overrides the existing value only. When a user navigates away from the component, and then back to the page again, the original value is used until it's changed again through PeopleCode.

The value for this property can be either of the following:

`Image.`*`imagename`*

or

`"Imagename"`

Oracle strongly recommends that if the PeopleCode program sets the LabelImage for an active button, it also sets the image appropriately if the button is disabled.

This property is read-write.

Example

```
Local Field &MyField;

&MyField = GetField(BUTTON_BOTTOM);
&MyField.LabelImage = Image.BTN_REFRESH;
```


LongTranslateValue

Description

This property returns a string that contains the Long translate (XLAT) value of the field if the field is based on a translate table.

If the field has a null value, a null string is returned. If the field isn't based on a translate table, or the value isn't in the translate table, the field's current value is returned. Because the current value can be of any type, this property has a type of Any.

Note. If you're accessing a field based on the Translate table, the Value property returns only the one or two letter XLAT value.

Use the ShortTranslateValue property to return the short translate value of a field.

This property is read-only.

Example

```
Local Any &VALUE;
Local Field &MYFIELD;

&MYFIELD = GetField();
&VALUE = &MYFIELD.LongTranslateValue;
If ALL(&VALUE) Then
   /* do processing */
End-if;
```

See Also

Chapter 18, "Field Class," Value, page 972 and Chapter 18, "Field Class," ShortTranslateValue, page 968

MessageNumber

Description

This property returns the error message number (as a number) if an error for this field is found after executing ExecuteEdits method. You can use this property in conjunction with the EditError property (which can be used to determine whether there are any errors) and the MessageSetNumber property (which contains the error message set number if an error is found.)

This property is read-only.

Example

```
For &I = 1 to &RECORD.FieldCount
   &MYFIELD = &RECORD.GetField(&I);
   If &MYFIELD.EditError Then
      &MSGNUM = &MYFIELD.MessageNumber;
      &MSGSET = &MYFIELD.MessageSetNumber;
      /* Do processing */
   End-If;
End-For;
```

MessageSetNumber

Description

This property returns the error message set number (as a number) if an error for this field is found after executing ExecuteEdits method. You can use this property in conjunction with the EditError property (which can be used to determine whether there are any errors) and the MessageNumber property (which contains the error message number if an error is found.)

This property is read-only.

Example

```
For &I = 1 to &RECORD.FieldCount
   &MYFIELD = &RECORD.GetField(&I);
   If &MYFIELD.EditError Then
      &MSGNUM = &MYFIELD.MessageNumber;
      &MSGSET = &MYFIELD.MessageSetNumber;
      /* Do processing */
   End-If;
End-For;
```

MouseOverMsgNum

Description

Use this property to override the message number for a message catalog mouse over popup page. Together with the MouseOverMsgSet property, this property uniquely identifies a Message Catalog entry to be used as the mouse over popup page.

The Message Text field from that Message Catalog entry becomes the title of the mouse over popup page; the Explanation field becomes the body of the mouse over popup page.

Important! For this property to have an effect, the page field definition must first be set as a message catalog mouse over popup in Application Designer. This property overrides the existing message number value only. When a user navigates away from the component, and then back to the page again, the original value is used until it's changed again through PeopleCode.

This property is read-write.

Example

For example, the following code could be added to the page Activate event to override the message set and number defined in Application Designer for the popup page associated with the DEBUG_PEOPLECD.DEBUG_CODE field.

```
GetField(DEBUG_PEOPLECD.DEBUG_CODE).MouseOverMsgSet = 2;
GetField(DEBUG_PEOPLECD.DEBUG_CODE).MouseOverMsgNum = 13;
```

See Also

Chapter 18, "Field Class," MouseOverMsgSet, page 964

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Using Page Controls," Enabling Message Catalog Pop-up Pages

MouseOverMsgSet

Description

Use this property to override the message set number for a message catalog mouse over popup page. Together with the MouseOverMsgNum field property, this property uniquely identifies a Message Catalog entry to be used as the mouse over popup page.

Important! For this property to have an effect, the page field definition must first be set as a message catalog mouse over popup in Application Designer. This property overrides the existing message set value only. When a user navigates away from the component, and then back to the page again, the original value is used until it's changed again through PeopleCode.

This property is read-write.

See Also

Chapter 18, "Field Class," MouseOverMsgNum, page 963

Name

Description

This property returns the name of the field definition that the field object is based on as a string value.

This property is read-only.

Example

```
WinMessage("The character field's name is : " | &CHARACTER.Name);
```

OriginalValue

Description

This property returns the value of a field, that is, the value that from the database. If the value hasn't been changed and saved by the user, it is the original value from the database. If the value has been changed and saved by the user, it is the existing value in the database.

Note. This property does *not* work for derived records. Original values are the database values, and derived records do not have a corresponding database value.

This property is read-only.

Example

```
&Orig = &MyField.OriginalValue;
If &Orig = &Date Then
   /* do current day processing */
Else
   /* do other processing */
End-If;
```

ParentRecord

Description

This property returns a reference to the record object for the record containing the field.

This property is read-only.

Example

```
&NUMBER_OF_FIELDS = &CHARACTER.ParentRecord.Fieldcount;

/* note that FieldCount is a property of the Record class */
```

PromptTableName

Description

This property returns the name of the prompt table (if any) associated with this field. This property returns a string value. This property returns Null if no prompt table is associated with the field.

This property is read-only.

SearchDefault

Description

If this property is set to True, system defaults (the default values set in the record field definitions) are enabled on search dialogs for the field. Setting this property to True *does not* cause the FieldDefault event to fire.

The system default is done only once, when the search dialog first starts, immediately after any SearchInit PeopleCode. If the user subsequently blanks out a field, the field isn't reset to the default value. Setting SearchDefault to False disables default processing for the field executing the property.

SearchDefault is effective only when used in SearchInit PeopleCode programs.

This property is read-write.

Example

```
&CHARACTER.SearchDefault = True;
```

```
/* assuming &CHARACTER is a search key, and has a default value */
```

This example turns on edits and system defaults for the SETID field in the search dialog box:

```
&SETID = GetField(INV_ITEMS.SETID);
&SETID.SeachDefault = True;
&SETID.SearchEdit = True;
```

SearchEdit

Description

If this property is set to True, SearchEdit enables system edits (edits specified in the record field definition) for the field, for the life of the search dialog box. Setting SearchEdit to False disables system edits. In the Add mode search dialog box, the following edits are performed when the end-user clicks the Add button. In any other mode, the following edits are performed when the end-user clicks the Search button:

- Formatting

- Required Field
- Yes/No Table
- 1/0 Table
- Translate Table
- Prompt Table

SearchEdit does not cause the FieldEdit, FieldChange, or SaveEdit PeopleCode events to fire during the search dialog.

You might use SearchEdit to control access to the system. For example, you can apply this function to the SETID field of a dialog box and require the user to enter a valid SETID before they are able to click OK on the search dialog box.

This property is read-write.

Considerations Using SearchEdit

If you use this method in the SearchInit event, the search page options are limited to the "=" and "IN" operators.

Example

```
&CHARCTER.SearchEdit = True;

/* assuming &CHARACTER is a search key, and contains an edit table
such as translate table defined in its field properties.  */
```

This example turns on edits and system defaults for the SETID field in the search dialog box:

```
&SETID = GetField(INV_ITEMS.SETID);
&SETID.SeachDefault = True;
&SETID.SearchEdit = True;
```

SetComponentChanged

Description

This property determines whether a change to a field (using PeopleCode) marks the component as changed and the data written to the database if the field is a database field.

This property takes a Boolean value: true, changes to the field mark the component as changed, false, the component is treated as if unchanged. The default value is true for database fields and false for derived fields.

The Set Component Changed checkbox for a page field in Application Designer determines if a user's action on a page marks the component as changed. For example, if Set Component Changedfor an edit box control is cleared in Application Designer, then user's editing on the edit box does not mark the component as changed and no save warning message is displayed if the user leaves the page without saving. The SetComponentChanged property for a field determines if a change on the field value using PeopleCode marks the component as changed and a save warning is issued when the user tries to exit the page without saving.

This property is only applicable after the Page Activate event has run. If you try to use this property before or during Page Activate, the user does not receive a warning if data has been changed.

This property is read-write.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Page Definitions," Setting Use Properties

Chapter 36, "Rowset Class," SetComponentChanged, page 2169

ShortTranslateValue

Description

This property returns a string that contains the Short translate (XLAT) value of the field if the field is based on a translate table.

If the field has a null value, a null string is returned. If the field isn't based on a translate table, or the value isn't in the translate table, the field's current value is returned. Because the current value can be of any type, this property has a type of Any.

Note. If you're accessing a field based on the translate table, the Value property returns only the one or two letter XLAT value.

Use the LongTranslateValue property to return the long translate value of a field.

This property is read-only.

Example

```
Local Any &VALUE;
Local Field &MYFIELD;

&MYFIELD = GetField();
&VALUE = &MYFIELD.ShortTranslateValue;
If ALL(&VALUE) Then
   /* do processing */
End-if;
```

See Also

Chapter 18, "Field Class," Value, page 972 and Chapter 18, "Field Class," LongTranslateValue, page 962

ShowRequiredFieldCue

Description

With PeopleTools 8, an asterisk (*) is displayed on pages beside fields that are defined as Required in Application Designer. You can use this property to specify whether this asterisk, also called the *required field cue,* is displayed for a particular field.

For example, many fields are made required or non-required either procedurally or through PeopleCode. This means they aren't defined as Required in Application Designer, and the end-user may be confused. For these fields, you can use this property.

Note. This property affects only fields where a required field cue is otherwise permissible. That is, regardless of the setting of the property, no cue is ever shown on a pushbutton, a display-only field, and so on.

This property is read-write.

SmartZero

Description

Use this property to specify if FieldChange PeopleCode programs are run when a user changes a zero to a blank, or a blank to a zero, for a field on a page.

This property takes a Boolean value: true, run FieldChange PeopleCode programs, false, do not. The default value is false.

When you set this property to true, in addition to treating a user change of blank-to-zero and zero-to-blank as a field change, the application server also adjusts the DisplayZero property for the runtime field object to display a blank or zero as entered by the user.

This property is read-write.

See Also

Chapter 18, "Field Class," DisplayZero, page 950 and Chapter 18, "Field Class," DisplayZeroChanged, page 950

SqlText

Description

This property is valid only for fields that have a dynamic view as their prompt record. If you set SqlText to a non-null value, that text is used instead of the dynamic view's normal text used for prompting.

Important! Even if the SQL text has been set in Application Designer, this property returns a blank string if you use the property before you've explicitly set it in PeopleCode. This property overrides the existing value only. When a user navigates away from the component, and then back to the page again, the original value is used until it's changed again through PeopleCode.

Suppose you wanted to have a different prompt table depending on the settings of other fields in the row. Normally you could use %EDITTABLE to dynamically specify the prompt table you want. However in this case there are too many possible combinations of values, which would require too many views. Furthermore, the values are customizable by the end-user or the application, which means even if you, the developer, wanted to, you couldn't provide all the combinations of views necessary. However you can generate the desired SQL text for the view in PeopleCode based on what the user enters.

If you use a dynamic view as the prompt table, and have the dynamic view contain a SQL object that is updated from PeopleCode, you could achieve this functionality. However, a SQL object is a shared object, so if multiple users used the same page, they overwrite each other's settings and the SQL object contains the SQL for the most recent user. Similarly if a single user had multiple rows on a page, the SQL object is valid only for the most recent row. This means if the user went to another row and did a prompt, they would get the wrong values again.

The purpose of this property is to enable you to specify the generated SQL text independently for each occurrence in each transaction. It enables you to override the text of a dynamic view being used as a prompt table on a field by field basis.

It is up to the developer to verify that the text specified for this property is valid, that is, that it selects the correct number of fields for the record definition, and so on.

This property is read-write.

StoredFormat

Description

This property returns the custom character format (as a string) for the field executing the property.

If the field doesn't have a custom format associated with it, the user receives a runtime error message.

If the field has a display format associated with it, you can change that using the DisplayFormat property.

This property is read-only.

Example

```
WinMessage("The character field's custom stored format is : " | &CHARACTER.Stored⇒
Format);

/* this assumes that &CHARACTER is a custom formatted */
/* field */
```

See Also

Chapter 18, "Field Class," DisplayFormat, page 948

Style

Description

If a page has a style sheet associated with it, this property can be used to specify a different style class with a field. Use this property to override the existing style class for a field.

Important! Even if the style class has been set in Application Designer, this property returns a blank string if you use the property before you've explicitly set it in PeopleCode. This property overrides the existing value only. When a user navigates away from the component, and then back to the page again, the original value is used until it's changed again through PeopleCode.

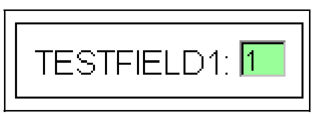
This property takes a string value.

This property is read-write.

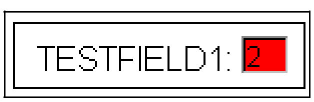
Example

The following example associates a test field first with a style class depending on the value of the field.

```
Local Field &field;

&field = GetField();

If TESTFIELD1 = 1 Then;
   &field.Style = "PSHYPERLINK";
End-If;

If TESTFIELD1 = 2 Then;
   &field.Style = "PSIMAGE";
End-If;
```



Field with PSHYPERLINK style



Field with PSIMAGE style

Type

Description

This property returns the type of field. The values can be one of the following strings:

- CHAR
- DATE
- DATETIME
- IMAGE (for static images)
- IMAGEREFERENCE
- LONGCHAR
- NUMBER
- SIGNEDNUMBER
- TIME

Note. Fields of type Attachment have a type of IMAGE.

This property is read-only.

Example

```
If &CHARACTER.Type = "NUMBER" Then
   /* perform processing */
Else
   /* error processing */
End-If;
```

Value

Description

This property contains the current value of the field, converted to an appropriate PeopleCode data type.

In most contexts, the Value property can be used to assign a new value to the field. However, there are some restrictions:

- You cannot assign the Value property in any RowSelect PeopleCode program.
- You cannot assign the current record field value to the Value property in any FieldEdit PeopleCode program.

Note. If you're accessing a field based on the translate table, Value returns only the one letter XLAT value. To get the full XLAT value, use the LongTranslateValue or ShortTranslateValue properties.

This property is read-write.

Considerations Using INSTALLATION or OPTIONS Tables

When using the INSTALLATION or OPTIONS table, you cannot use the Value property. You must use the old style format, not the field object format. For example, the following code is invalid:

```
If %Page = Page.GP_RUN_TYPE And
      INSTALLATION.TL.Value = "N" Then
   &RS2.HideAllRows();
```

While the following code is valid:

```
If %Page = Page.GP_RUN_TYPE And
      INSTALLATION.TL = "N" Then
   &RS2.HideAllRows();
```

Considerations Using Character Values

Previously, if you had an edit-box field, and if the end-user selected the value in it and deleted the value, leaving the field empty, the value of the field in PeopleCode was not an empty (zero-length) string.

Now, the value of such a field *is* an empty (zero-length) string.

In addition, if the user adds one or more space characters to a field, the field still returns a Null string.

The following is how to check for this:

```
If (fieldname.Value = "") Then
```

Example

```
&CHARACTER.Value = "Hello";
```

See Also

Chapter 18, "Field Class," LongTranslateValue, page 962 and Chapter 18, "Field Class," ShortTranslateValue, page 968

Visible

Description

This property is True if this field is visible in the page displaying it. Setting this property to False hides the field. Because every field is implicitly associated with a rowset, row, and record, setting the Visible property for a field on the first page of a component hides only that field. If that field is repeated on other pages in the component, the other occurrences of the field aren't hidden.

This property is read-write.

Example

The following code hides the field &ROUND_OPTION based on the value of AVG_DFLT_OPTION:


```
Local field &ROUND_OPTION;

&ROUND_OPTION = GetField();

If AVG_DFLT_OPTION = "Y" Then
   &ROUND_OPTION.SetDefault();
   &ROUND_OPTION.Visible = True;
Else
   &ROUND_OPTION.Visible = False;
End-If;
```

Chapter 19

File Class

This chapter describes the File class, which provides methods and properties for reading from and writing to external files and discusses the following topics:

- Data type of a file object
- Scope of a file object
- File security considerations
- File access interruption recovery
- Plain text files
- Automatic PeopleCode generation
- File layout error processing
- File class built-in functions
- File class methods
- File class properties
- File layout examples
- Multiple file layouts

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Constructing File Layouts and Performing Data Interchanges"

Understanding File Layout

PeopleTools supports reading and writing to plain text files, and to files that have a format based on a File Layout definition that has been created in Application Designer.

- If the file is a plain text file, data is read or written using text strings.
- If the file is based on a File Layout, you can use text strings, rowset, or record objects.

This simplifies reading, writing, and manipulating hierarchical transaction data with PeopleCode.

File layout methods and properties are noted as such in their descriptions.

See Also

Chapter 19, "File Class," Plain Text Files, page 978

Chapter 19, "File Class," File Layout Examples, page 1016

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Constructing File Layouts and Performing Data Interchanges"

Data Type of a File Object

The File Object is an instance of the File class. A file object is declared using the File data type. For example,

```
Local File &MYFILE;
```

This creates an object &MYFILE of the class File.

Scope of a File Object

A file object can only be instantiated from PeopleCode. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

A file object is passed to and returned from PeopleCode functions and methods as a reference, so the file object is never copied; rather, alternate identifiers are used to refer to the same object. Any change made to a file using one identifier has the same effect as it would with any other identifier for that file.

In the following code example, two variables, &F1 and F2, are declared by using the data type File. The file is opened using the GetFile built-in function. Next, &F1 is assigned to &F2. This does *not* copy &F1 to &F2. Instead, &F1 and &F2 both refer to the same object. Therefore, in the last step when &F2 is closed, &F1 is closed too.

```
Local File &F1, &F2;

&F1 = GetFile("somefile.txt", "R");
If &F1.IsOpen Then
   &F2 = &F1;
   &F2.Close();  /* Now &F1 is also closed. */
End-if;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

File Security Considerations

When you're using file objects in PeopleCode that might run on a server, you must be aware of some security concerns. The underlying system doesn't provide security checks on access to the files. Instead, the PeopleCode programs use whatever authority the server process has, and *not* that of the user. This means you must write your PeopleCode program to prevent the user from reading or writing files that they should not. In particular, be especially wary of opening files where any part of the filename is derived from user input.

File Access Interruption Recovery

You can use the GetPosition or SetPosition methods to minimize the loss of work in the event of a system failure during file access. To use them and recover from access interruptions, you must access the file in Update mode using the GetFile built-in function or the Open method, specifying the mode parameter "U". You need to use this mode in anticipation of possible interruptions if you want to recover from them later.

To start reading or writing from the beginning of a file, use SetPosition to set a read/write position of 0, immediately after you open the file in Update mode.

Warning! In Update mode, any write operation clears the file of all data that follows the position you set.

When reading from or writing to a file, use GetPosition periodically to determine your current byte position in the file. This establishes a checkpoint to which you can return after a failure. You must save the checkpoint value in a separate file or a database so you can retrieve it later. How often you save a checkpoint depends on your requirements: the less work you want to redo, the more frequent your checkpoints should be. You may also want to save information identifying the data you're reading or writing at the time.

After a failure interrupts your file access, reopen the file in Update mode. You can then retrieve the last checkpoint value you saved, and use SetPosition to apply that value. Your read/write position in the file is the position of the last checkpoint, and you can continue reading or writing from there.

Use the Update mode with GetPosition and SetPosition only for recovering from interruptions as described here, or for starting at the beginning of the file again.

Note.

For fixed-width file layouts, the field start positions and lengths are computed in 8-bit bytes, except in the case of UCS2 or UTF-16 where they are computed in 16-bit double-bytes. This may create complications when using variable width encodings (for example, UTF-8 or Shift-JIS), when the data is elsewhere calculated in characters (rather than bytes), because the data involved may contain characters of varying byte-width.

See Also

Chapter 19, "File Class," Open, page 989

Chapter 19, "File Class," SetPosition, page 1000

Chapter 19, "File Class," GetPosition, page 986

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

Plain Text Files

To read and write from plain text files involves reading and writing strings of data. These text strings can be manipulated with built-in string functions, like RTrim, Find, Replace, and so on.

Note. If your data is hierarchical in nature, or based on existing PeopleSoft records or pages, use a File Layout definition for reading and writing your data, rather than doing it line by line (or field by field.)

The following example creates an array of array of string, then reads in two files, one into each "column" of the array. The Names file contains names; the Numbers file contains employee numbers. The ReadLine method reads each successive line in a file, until it reaches the end of the file.

Notice that the first file is opened using the GetFile function. The second file is not opened using GetFile, but rather with the Open method. After the data is read into the array, you can do processing on the data. The end of the program writes the changes back to the files, using the WriteLine method, which includes a system end of line character at the end of every line.


```
Local array of array of string &BOTH;
Local File &MYFILE;
Local string &HOLDER;

/* Create empty &BOTH array */
&BOTH = CreateArrayRept(CreateArrayRept("", 0), 0);

/* Read first file into first column */

&MYFILE = GetFile("names.txt", "R");
While &MYFILE.ReadLine(&HOLDER);
   &BOTH.Push(&HOLDER);
End-While;

/* read second file into second column */

&MYFILE.Open("numbers.txt", "R");
&LINENO = 1;
While &MYFILE.ReadLine(&HOLDER);
   If &LINENO > &BOTH.Len Then
      /* more number lines than names, use a null name */
      &BOTH.Push(CreateArray("", &HOLDER));
   Else
      &BOTH[&LINENO].Push(&HOLDER);
   End-If;
   &LINENO = &LINENO + 1;
End-While;

/* if more names than numbers, add null numbers */
For &LINENO = &LINENO to &BOTH.Len
      &BOTH[&LINENO].Push("");
End-For;
&MYFILE.Close();
/* do processing with array */
/* write data back to files */

&MYFILE1 = GetFile("names.txt", "A");
&MYFILE2 = GetFile("numbers.txt", "A");

/* loop through array and write to files */

For &I = 1 To &BOTH.Len
   &STRING1 = &BOTH[&I][1];
   &MYFILE1.writeline(&STRING1);
   &STRING2 = &BOTH[&I][2];
   &MYFILE2.writeline(&STRING2);
End-For;

&MYFILE1.Close();
&MYFILE2.Close();
```

See Also

Chapter 19, "File Class," File Layout Examples, page 1016

Automatic PeopleCode Generation

After you create a File Layout definition, you can use PeopleCode to access it. This PeopleCode can be long and complex. Rather than write it directly, you can drag and drop the File Layout definition from Application Designer Project View into an open PeopleCode edit pane. This is primarily used for importing data. Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify to meet your requirements.

The following is just a snippet of the code that is generated:

```
Function EditRecord(&REC As Record) Returns boolean ;
   Local integer &E;
REM   &REC.ExecuteEdits(%Edit_Required + %Edit_DateRange + %Edit_YesNo + %Edit_⇒
TranslateTable + %Edit_PromptTable + %Edit_OneZero);
   &REC.ExecuteEdits(%Edit_Required + %Edit_DateRange + %Edit_YesNo + %Edit_One⇒
Zero);
   If &REC.IsEditError Then
      For &E = 1 To &REC.FieldCount
         &MYFIELD = &REC.GetField(&E);
         If &MYFIELD.EditError Then
            &MSGNUM = &MYFIELD.MessageNumber;
            &MSGSET = &MYFIELD.MessageSetNumber;
            &LOGFILE.WriteLine("****Record:" | &REC.Name | ", Field:" |⇒
 &MYFIELD.Name );
            &LOGFILE.WriteLine("****" | MsgGet(&MSGSET, &MSGNUM, ""));
         End-If;
      End-For;
      Return False;
   Else
      Return True;
   End-If;
End-Function;
.
.
.
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Constructing File Layouts and Performing Data Interchanges"

End Of Line Considerations

Input files for file layouts of type fixed or CSV must not contain embedded CR/LF within the field data.

However, input files for file layouts of type XML can contain embedded CR/LF within field data for a character type file field that is the associated with a long-character type database field that has unlimited length—that is, the maximum length specified as 0.

File Layout Error Processing

If an error occurs on any field in any record of a rowset object populated with the ReadRowset method, the rowset object's property IsEditError returns True. For example, you can use the method ExecuteEdits on a record, to verify that the data in the record is valid (has the correct format, the right data type, and so on.) This type of error is indicated by the IsEditError.

In some instances, however, the rowset object won't receive the error; in that case the file object's IsError property returns True. To discover all field errors, check both properties after executing ReadRowset.

To determine which field has the error, you must examine the EditError property of every field in the rowset to find the one returning True. You can then examine that field's MessageSetNumber and MessageNumber properties to determine the relevant error message. The following example shows how this might be done:

```
&MYFILE.Open(&SOMENAME, "R");
&MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT);
&MYROWSET = &MYFILE.ReadRowset();
If &MYFILE.IsError Then
  For &I = 1 to &MYROWSET.ActiveRowCount
    If &MYROWSET.GetRow(&I).IsEditError Then
      &ROW = &MYROWSET.GetRow(&I);
      For &J = 1 to &ROW.RecordCount
        If &ROW.GetRecord(&J).IsEditError Then
          &REC = &ROW.GetRecord(&J);
          For &K = 1 to &REC.FieldCount
            If &REC.GetField(&K).EditError Then
              /* Examine the field&rsquo;s
                 MessageSetNumber and MessageNumber properties,
                 and respond accordingly */
         End-If;
          End-For;
      End-If;
      End-For;
    End-If;
  End-For;
End-If;

&MYFILE.Close();
```

Note. Only field errors set the IsError, IsEditError, or EditError properties. All other errors triggered by File class methods terminate the PeopleCode program.

See Also

Chapter 19, "File Class," IsError, page 1010

Chapter 19, "File Class," ReadRowset, page 994

Chapter 19, "File Class," IgnoreInvalidId, page 1010

Chapter 36, "Rowset Class," IsEditError, page 2166

Chapter 34, "Record Class," ExecuteEdits, page 2086

Chapter 25, "Message Classes," ExecuteEdits, page 1268

Chapter 18, "Field Class," MessageNumber, page 962

Chapter 18, "Field Class," MessageSetNumber, page 963

Working With Relative Paths

Relative paths apply to the following PeopleCode operations: the Open method of the File class and the CreateDirectory, FileExists, FindFiles, GetFile, GetTempFile, and RemoveDirectory built-in functions

If you specify a relative path, that path is appended to the path constructed from a system-chosen environment variable. Environment variables are checked in a particular order to find one that is set. This means if the first environment variable in order is found to be set, that's the one that is used. If the environment variable isn't found to be set, the next one is checked, and if found, is used, and so on.

If the PeopleCode operation is called from a program running in an environment started through psadmin (for example, within an application server domain), the PeopleCode operation checks in the following order to determine whether an environment variable is set:

1. *PS_FILEDIR*
2. *PS_SERVDIR*

If the PeopleCode operation is called from an environment *not* started through psadmin (for example, a stand-alone call to Application Engine), the PeopleCode operation checks in the following order to determine whether an environment variable is set:

1. *PS_FILEDIR*
2. *PS_SERVDIR*
3. *TEMP*

In both cases, the PeopleCode operation then uses the value of the first environment variable that is found to be set and appends the specified relative path to it in order to determine the resulting full path that will be used to locate the file. The resulting path is shown in the following table:

Environment Variable	***Resulting Path***
PS_FILEDIR	*PS_FILEDIR**relative_path*
PS_SERVDIR	*PS_SERVDIR***files***relative_path*
TEMP	*TEMP**relative_path*

If the PeopleCode operation is not successful at the specified location, it will return Null or False (as appropriate for the invoking operation) and it will *not* attempt to use the value of one of the other environment variables.

Note. In the preceding examples, the Windows directory separator, a backslash, was used. For UNIX directories, the directory separator is a forward slash.

Note. The PS_FILEDIR environment variable is *not* initialized and set to a path automatically. If you want to use this environment variable, you must set it yourself.

Note. Your system security should verify if a user has the correct permissions before allowing access to a drive. (For example, if a user changes their TEMP environment variable to a network drive they don't normally have access to, your system security should detect this.)

File Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateDirectory

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," FileExists

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," FindFiles

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetTempFile

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," RemoveDirectory

File Class Methods

In this section, we discuss the File class methods. The methods are discussed in alphabetical order.

Close

Syntax

Close()

Description

The Close method discards any changes that haven't been written to the external file, disassociates the file object from the external file, releases all resources connected with the file object, and causes the file object to go out of scope.

You cannot use any methods or properties on the object after it is closed. You must get another file object (using the GetFile function) and instantiate another file object after you use Close.

Parameters

None.

Returns

None.

Example

```
&MYFILE.Open("somefile.txt", "W", %FilePath_Relative);
&MYFILE.WriteLine("Some text.");
&MYFILE.Close();
```

See Also

Chapter 19, "File Class," Open, page 989 and Chapter 19, "File Class," IsOpen, page 1012

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

CreateRowset

Syntax

CreateRowset()

Description

The CreateRowset method is a file layout method. It instantiates a PeopleCode rowset object containing one unpopulated row, based on the file layout definition specified with SetFileLayout.

Note. You must specify a file layout using the SetFileLayout method before using CreateRowset, otherwise it returns NULL.

See Chapter 19, "File Class," SetFileLayout, page 998.

After the empty rowset object has been created, you can use Rowset class methods such as Select or Fill, and built-in functions such as GetLevel0 or GetRowset, to populate the rowset with data. After the rowset contains data, you can use the method WriteRowset to write the data to a file.

Don't use CreateRowset when reading from a file. Instead, use the ReadRowset method to instantiate and populate rowsets with data from the file.

Parameters

None.

Returns

An empty rowset object.

Example

```
&SOMEROWSET = &MYFILE.CreateRowset();
```

The following rowset is created from a file object, then populated with data using the GetLevel0 function:

```
&FILEROWSET = &MYFILE.CreateRowset();
&FILEROWSET = GetLevel0();
&MYFILE.WriteRowset(&FILEROWSET, True);
```

The following rowset is created from a file object, then populated with data using Fill method:

```
&FILEROWSET = &MYFILE.CreateRowset();
&NUM_READ = &FILEROWSET.Fill("where MYRECORD = :1", &UVAL);
```

See Also

Chapter 19, "File Class," ReadRowset, page 994; Chapter 19, "File Class," SetFileLayout, page 998 and Chapter 19, "File Class," WriteRowset, page 1006

Chapter 36, "Rowset Class," page 2127

Delete

Syntax

```
Delete()
```

Description

Use the Delete method to delete an open file. You cannot delete a closed file.

Parameters

None.

Returns

None.

Example

```
/* Open a temporary file. Use the "N" parameter to ensure
the file is new. */

&MyFile = GetFile("temp.txt", "N");

/* verify the file was instantiated successfully */

/* do processing using the temporary file */

/* delete the temporary file */

&MyFile.Delete();
```

See Also

Chapter 19, "File Class," Open, page 989 and Chapter 19, "File Class," Close, page 984

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

GetPosition

Syntax

```
GetPosition()
```


Description

The GetPosition method returns the current read or write position in the external file. GetPosition works only with a file that was opened in Update mode. This method is designed to be used in combination with the SetPosition method to establish checkpoints for restarting during file access.

Note. Currently, the effect of the Update mode and the GetPosition and SetPosition methods is not well defined for Unicode files. Use the Update mode only on files stored with a non-Unicode character set.

Parameters

None.

Returns

A number representing the current read or write position in the file.

Note. When you use ReadRowset to read from a file in Update mode, the CurrentRecord property returns, the entire record just read. The current read/write position is at the *end* of the CurrentRecord, that is, just past the end of the rowset.

Example

The following example opens a file in Update mode, and saves the current position after each read operation:

```
&MYFILE.Open(&SOMENAME, "U");
If &MYFILE.IsOpen Then
   while &MYFILE.ReadLine(&SOMESTRING)
      &CURPOS = &MYFILE.GetPosition();
      /* Save the value of &CURPOS after each read,
         and process the contents of each &SOMESTRING */
   End-While;
End-If;
&MYFILE.Close();
```

See Also

Chapter 19, "File Class," Open, page 989 and Chapter 19, "File Class," SetPosition, page 1000

Chapter 19, "File Class," File Access Interruption Recovery, page 977

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

GetString

Syntax

`GetString([Strip_Line_Terminator])`

Description

Use the GetString method to return the entire file as a single string.

Note. After this method completes successfully, the original file is deleted.

You can specify whether the resulting string is to include the line terminator characters or not by using the *Strip_Line_Terminator* parameter. The default value for this parameter is false, which means the resulting string includes the line terminator characters at the end of each line.

For example on a Unix system, with a line terminator of a LF, the resulting string includes not only the data for each line, but the LF character as well.

Parameters

Parameter	***Description***
Strip_Line_Terminator	Specify whether the line terminators are to be stripped or not.

Returns

A single string containing the entire contents of the file.

Example

The following example creates a file on a Windows system, then retrieves it as a single line of text.

Note. The file is destroyed on successful completion of this method.

```
Local File &File = GetFile("c:\temp\junk\something.txt", "W", %FilePath_Absolute)
/* write a bunch of > 2048 length lines */

Local string &piece, &input;
Local integer &I;

&piece = "123456789-";
While Len(&piece) < 2048
   &piece = &piece | &piece;
End-While;

&File.WriteString(&piece);
&File.WriteString(&piece);
&File.WriteString(&piece);
&input = &File.GetString( True);
&File.Close();
Local string &pieces = &piece | &piece | &piece;

/* Note that the result of this message should indicate &pieces is the same as⇒
 &input */
MessageBox(0, "", 0, 0, "&piece = &input: Len(&pieces)=" | Len(&pieces) | " Len⇒
(&input)=" | Len(&input) | " Same? " | (&pieces = &input));
```

See Also

Chapter 19, "File Class," ReadLine, page 993; Chapter 19, "File Class," WriteLine, page 1002 and Chapter 19, "File Class," WriteRaw, page 1003

Open

Syntax

Open*(filespec,mode* [, *charset*] [, *pathtype*])

Description

The Open method associates the file object with an external file for input or output.

If the File object is currently associated with a file, that file is closed first. In addition, any file opened for writing (by a call to the GetFile function or the Open method) by a PeopleCode program that runs in the Process Scheduler is automatically managed by the Report Repository.

You can use the GetFile or GetTempFile functions to access an external file, but each execution of GetFile or GetTempFile instantiates a new file object. If you plan to access only one file at a time, you need only one file object. Use GetFile or GetTempFile to instantiate a file object for the first external file you access. Then, use Open to associate the same file object with as many different external files as you want. However, if you expect to have multiple files open at the same time, you need to instantiate multiple file objects with GetFile or GetTempFile.

GetFile and Open both perform implicit commits. Therefore, the GetTempFile function has been introduced specifically to avoid these implicit database commits. GetTempFile differs from GetFile in two respects:

- GetTempFile does not perform an implicit commit.

- GetTempFile does not make the associated file available through the Report Repository even when the calling PeopleCode program is run through the Process Scheduler.

Therefore, GetTempFile can be a good choice when you wish to avoid implicit database commits and when you do not need to have the file managed through the Report Repository. Otherwise, GetTempFile operates exactly the same as GetFile.

Parameters

Parameter	***Description***
filespec	Specify the name, and optionally, the path, of the file you want to open.
mode	A string indicating the manner in which you want to access the file. The mode can be one of the following: "R" (Read mode): opens the file for reading, starting at the beginning. "W" (Write mode): opens the file for writing. ***Warning!*** When you specify Write mode, any existing content in the file is discarded and is overwritten. "A" (Append mode): opens the file for writing, starting at the end. Any existing content is retained. "U" (Update mode): opens the file for reading or writing, starting at the beginning of the file. Any existing content is retained. Use this mode and the GetPosition and SetPosition methods to maintain checkpoints of the current read/write position in the file. In Update mode, any write operation clears the file of all data that follows the position you set. **Note.** Currently, the effect of the Update mode and the GetPosition and SetPosition methods is not well defined for Unicode files. Use the Update mode only on files stored with a non-Unicode character set. "E" (Conditional "exist" read mode): opens the file for reading only if it exists, starting at the beginning. If it doesn't exist, Open has no effect. Before attempting to read from the file, use the IsOpen property to confirm that it's open. "N" (Conditional "new" write mode): opens the file for writing, only if it doesn't already exist. If a file by the same name already exists, Open has no effect. Before attempting to write to the file, use the IsOpen property to confirm that it's open. You can insert an asterisk (*) in the filename to ensure that a new file is created. The system replaces the asterisk with numbers starting at 1 and incrementing by 1, and checks for the existence of a file by each resulting name in turn. It uses the first name for which a file doesn't exist. In this way you can generate a set of automatically numbered files. If you insert more than one asterisk, all but the first one are discarded.

Parameter	***Description***
charset	A string indicating the character set you expect when you read the file, or the character set you want to use when you write to the file. You can abbreviate Unicode UCS-2 to "U" and the host operating system's default non-Unicode (sometimes referred to as the ANSI character set) to "A". All other character sets must be spelled out in full, for example, ASCII, Big5, Shift-JIS, or UTF8. If "A" is specified as the character set, or you don't specify a character set, the character set used is dependent on the application server configuration. On a Windows application server, the default non-Unicode character set is dependent on the Windows ANSI Codepage (ACP) which can be checked using the DOS command chcp. On a Unix application server, the default non-Unicode character set is specified in the application server configuration file, psappsrv.cfg, and can be modified using PSADMIN. You can also use a record field value to specify the character set (for example, `RECORD.CHARSET`.) A list of supported character set names valid for this argument can be found in *PeopleTools 8.51 PeopleBook: PeopleSoft Global Technology*. See *PeopleTools 8.51 PeopleBook: Global Technology*, "Selecting and Configuring Character Sets and Language Input and Output," Character Sets in the PeopleSoft Pure Internet Architecture. **Note.** If you attempt to read data from a file using a different character set than was used to write that data to the file, the methods used generate a runtime error or the data returned is unusable. When a file is opened for reading using the "U" *charset* argument, Open expects the file to begin with a Unicode Byte Order Mark (BOM). This mark indicates whether the file is written in big endian order or little endian order. A BOM consisting of the hex value 0xFEFF indicates a big endian file, a BOM consisting of the hex value 0xFFEF indicates a little endian file. If the Unicode UCS-2 file being opened does not start with a BOM, an error is returned. The BOM is automatically stripped from the file when it is read into the buffers by Open. When a file is opened for writing using the "U" *charset* argument, the appropriate Unicode BOM is automatically written to the start of the file dependening on whether the application server hardware platform operates in little endian or big endian mode. Byte Order Marks are not expected or supported for files in other character sets, including UTF8. However, some applications allow or expect the UTF-8 sequence of the BOM (0xEF BB BF). In case the consuming application does expect the BOM for a UTF-8 file, the UTF8BOM character set is available as a Open character set to create a UTF-8 file with the BOM. **Note.** The UTF-8 sequence 0xEF BB BF, if misinterpreted as ISO-8859-1, appears as ISO characters ï»¿. When working with XML documents, specify UTF8 for *charset.* If you are writing an XML file using a different character set, you must remember to include a charset declaration in the XML file.

Parameter	Description
pathtype	If you have prepended a path to the file name, use this parameter to specify whether the path is an absolute or relative path. The valid values for this parameter are: • %FilePath_Relative (default) • %FilePath_Absolute If you don't specify *pathtype* the default is %FilePath_Relative. If you specify a relative path, that path is appended to the path constructed from a system-chosen environment variable. A complete discussion of relative paths and environment variables is provided in documentation on the File class. See Chapter 19, "File Class," Working With Relative Paths, page 982. If the path is an absolute path, whatever path you specify is used verbatim. You must specify a drive letter and the complete path. You can't use any wildcards when specifying a path. The Component Processor automatically converts platform-specific separator characters to the appropriate form for where your PeopleCode program is executing. On a Windows system, UNIX "/" separators are converted to "\", and on a UNIX system, Windows "\" separators are converted to "/". **Note.** The syntax of the file path does *not* depend on the file system of the platform where the file is actually stored; it depends only on the platform where your PeopleCode is executing.

Returns

None.

Example

The following example opens an existing Unicode file for reading:

```
&MYFILE.Open(&SOMENAME, "E", "U");
If &MYFILE.IsOpen Then
   while &MYFILE.ReadLine(&SOMESTRING)
      /* Process the contents of each &SOMESTRING */
   End-While;
   &MYFILE.Close();
End-If;
```

The following example opens a numbered file for writing in ANSI format, without overwriting any existing files:

```
&MYFILE.Open("C:\temp\item*.txt", "N", %FilePath_Absolute);
If &MYFILE.IsOpen Then
   &MYFILE.WriteLine("Some text.");
   &MYFILE.Close();
End-If;
```


See Also

Chapter 19, "File Class," IsOpen, page 1012; Chapter 19, "File Class," SetPosition, page 1000; Chapter 19, "File Class," GetPosition, page 986 and Chapter 19, "File Class," Close, page 984

Chapter 19, "File Class," File Access Interruption Recovery, page 977

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetTempFile

ReadLine

Syntax

```
ReadLine(string)
```

Description

The ReadLine method reads one line of text from the external file. The line includes the newline character, but ReadLine strips out the newline character and inserts the result into the string variable *string.*

When ReadLine is executed, it moves the starting point for the next read operation to the end of the text just retrieved, so each line in the file can be read in turn by subsequent ReadLine operations. When no more data remains to be read from the file, ReadLine returns False, and clears the string variable of any content.

Parameters

Parameter	***Description***
string	A string variable that receives the input text.

Returns

A Boolean value: True if the method succeeds, False otherwise. The return value is *not* optional, it is required.

Example

The following example reads a file called &MYFILE and puts each line as a separate element in an array.


```
Local File &MYFILE;
Local array of string &MYARRAY;
Local string &TEXT;

&MYFILE = GetFile("names.txt", "R");
&MYARRAY = CreateArrayRept("", 0);
While &MYFILE.ReadLine(&TEXT);
   &MYARRAY.Push(&TEXT);
End-While;
&MYFILE.Close();
```

See Also

Chapter 19, "File Class," GetString, page 988; Chapter 19, "File Class," ReadRowset, page 994; Chapter 19, "File Class," WriteString, page 1008 and Chapter 19, "File Class," WriteLine, page 1002

ReadRowset

Syntax

ReadRowset()

Description

The ReadRowset method is a file layout method. It instantiates a PeopleCode rowset object based on the file layout definition, then populates the rowset with one transaction from the file. A transaction is considered to be one instance of level zero data contained in a file record, plus all of its subordinate data. If you put more than one segment at level zero, each segment is read in sequence.

Note. You must specify a file layout using the SetFileLayout method before using ReadRowset, otherwise it will return NULL.

When ReadRowset is executed, it moves the starting point for the next read operation to the beginning of the next rowset, so each transaction in the file can be read in turn by subsequent ReadRowset operations. When no more data remains to be read from the file, ReadRowset returns NULL.

If you're using the SetFileId method with ReadRowset to process an input file based on multiple file layouts, ReadRowset returns NULL when it reads a FileId file record (line) between the rowsets.

When ReadRowset returns NULL, you can use the IsFileId property to determine if you've reached the end of the file or a FileId record.

Note. When using ReadRowset, if a value in the file exceeds the defined length in the file layout, it is ignored. The given record field is flagged with an edit error which can be programmatically checked.

If the ReadRowset encounters a line in the file containing the FileId, and the lines following this are *not* a new rowset, the process considers it to be an invalid FileId. You can specify whether to ignore the invalid record or terminate the PeopleCode with the IgnoreInvalidId property.

Note. If you're using the SetFileId method with ReadRowset to process an input file based on multiple layouts, FileId file records between the rowsets are considered to be valid file records, and won't generate any errors, regardless of the state of the IgnoreInvalidId property.

See Chapter 19, "File Class," SetFileLayout, page 998.

Considerations for Using Dates With ReadRowset

Single digits in dates in the form MMDDYY or MMDDYYYY must be padded with zeros. That is, if the date in your data is February 3, 2000, the form must be:

```
02/03/2000
```

or

```
02/03/00
```

The following is *not* valid.

```
2/3/00
```

Considerations for Using XML With ReadRowset

If all of the fields in the file layout are not present in the XML, no data is written to the database. If there are additional tags in the XML, they are ignored.

Considerations for Using Nested Data with ReadRowset

If you are processing input files with a large amount of nested data, your application server may run out of memory before the system finishes processing all of the data.

This may happen because of the differences between processing a single-level rowset and a multi-level rowset. If you are reading the rows of a single-level (level 0) rowset from a file, the file is processed one row at a time, that is, only one row resides in memory at a time.

However, if you are reading the rows of a multi-level (parent-child) rowset from a file, then for each level 0 row, *all* of the associated child rows (and all of their associated child rows) simultaneously reside in memory. As a result, during the processing a large input data file that is associated with a multi-level rowset (through a multi-level file layout definition), your application server may run out of memory.

To work around this, consider doing one of the following:

- Retain your original file layout definition and split the original input file into smaller (but structurally unchanged) pieces
- Flatten out your original file layout definition (that is, split it up into several single-level definitions) as well as split the original input file into several single-level pieces.

Considerations for Using Default Values with ReadRowset

The system variables related to date and time (for example, %Date, %Time, and %DateTime) cannot be used to specify the value of the Default Value property of a file layout field. This topic is covered in detail in the Application Designer PeopleBook.

See *PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide*, "Constructing File Layouts and Performing Data Interchanges," Specifying File Field Properties.

Parameters

None.

Returns

A populated rowset.

Example

The following example reads and processes an entire file. The data in the file is based on a single File layout definition:

```
&MYFILE.GetFile(&SOMENAME, "R");
If &MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT) then
   &SOMEROWSET = &MYFILE.ReadRowset();
   While &SOMEROWSET <> NULL
      /* Process the contents of each &SOMEROWSET */
      &SOMEROWSET = &MYFILE.ReadRowset();
   End-While;
End-If;
&MYFILE.Close();
```

See Also

Chapter 19, "File Class," CurrentRecord, page 1009; Chapter 19, "File Class," IgnoreInvalidId, page 1010; Chapter 19, "File Class," GetString, page 988; Chapter 19, "File Class," ReadLine, page 993; Chapter 19, "File Class," SetFileLayout, page 998; Chapter 19, "File Class," WriteRecord, page 1004 and Chapter 19, "File Class," WriteRowset, page 1006

Chapter 19, "File Class," Multiple File Layouts, page 1029

Chapter 19, "File Class," ReadRowset Example, page 1025

Chapter 36, "Rowset Class," IsEditError, page 2166

SetFileId

Syntax

`SetFileId`*`(fileid,position)`*

Description

The SetFileId method is a file layout method. If your input file contains data based on more than one File Layout definition, you must use this method in combination with the CurrentRecord and IsNewFileId properties and the ReadRowset method to process the file correctly.

Note. SetFileId, CurrentRecord and IsNewFileId don't apply to CSV and XML format input files. You can use only fixed format files to implement multiple file layouts.

See Chapter 19, "File Class," CurrentRecord, page 1009; Chapter 19, "File Class," IsNewFileId, page 1011 and Chapter 19, "File Class," ReadRowset, page 994.

At each point in the input file where the structure of the rowset changes, there must be an extra line in the file containing the file record (line), the FileId file record (line), that signifies that the change. Each FileId file record must have the following components:

- A value that designates it as a Fileid. Only one FileId value is necessary throughout the file, such as "999".
- A name field that identifies the file layout needed for the rowset that follows. This field could contain the name of the file layout as it's defined in Application Designer.

These lines containing the FileId are *not* part of any rowset. They can contain other information, which will be disregarded by the system. The FileId identifier and the file layout names aren't automatically stored anywhere; they exist only in the input file's FileId file records and in the PeopleCode.

To process an input file that requires multiple file layouts:

1. Use SetFileLayout to specify the file layout definition to use.

   If you're specifying the initial file layout for this file, it doesn't have to be the correct one. You can determine the correct file layout to use for the first rowset during a subsequent step.

2. Use SetFileId to specify the value of the FileId field, *fileid*, that's used throughout the file to designate FileId file records.

   Note. After each execution of SetFileLayout, the SetFileId setting is disabled. Be sure to re-specify the FileId value if you expect the file layout to change, so the system continues looking for the next FileId file record.

   ---

3. Use ReadRowset to read the next rowset from the file.

   The current file record is also stored in the CurrentRecord property as a string. The PeopleCode process determines whether it's the beginning of a new rowset, or a FileId file record according to the *fileid* you specified. If it's a FileId file record, the process sets the IsNewFileId property to True; if not, it sets the IsNewFileId property to False.

   Note. If this is the first execution of ReadRowset on the file, it returns NULL upon encountering the initial FileId file record, but still stores that file record in CurrentRecord and sets IsNewFileId accordingly.

   ---

4. If the rowset returned isn't NULL, process that rowset as necessary.

5. If IsNewFileId is False, go back to step 3 to continue reading rowsets from the file. If IsNewFileId is True, examine CurrentRecord to determine which new file layout to use, and go back to step 1.

You can repeat this procedure one rowset at a time, proceeding through the remainder of the input file. When ReadRowset returns NULL, no more rowset data is available.

Parameters

Parameter	***Description***
fileid	Specify the value of the FileId field used in the current file.
position	Specify the column in the FileId file record where the FileId field starts. The default is 1.

Returns

None.

Example

```
&rsFile = &MYFILE.ReadRowset();
&CurrentRecord = &MYFILE.CurrentRecord;
&IsNewFileID = &MYFILE.IsNewFileId;
If &MYFILE.IsNewFileId Then
   &FILELAYOUT = FindFileID(&CurrentRecord);
   &MYFILE.SetFileLayout(@("FileLayout." | &FILELAYOUT));
   &MYFILE.SetFileId("999", 1);
End-If;
```

See Also

Chapter 19, "File Class," CurrentRecord, page 1009; Chapter 19, "File Class," IsNewFileId, page 1011; Chapter 19, "File Class," ReadRowset, page 994 and Chapter 19, "File Class," SetFileLayout, page 998

Chapter 19, "File Class," File Layout Examples, page 1016

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Constructing File Layouts and Performing Data Interchanges"

SetFileLayout

Syntax

SetFileLayout(FILELAYOUT.*filelayoutname*)

Description

The SetFileLayout method is a file layout method. It associates a specific file layout definition with the file object executing this method, providing easy access to rowset data. The file object must be associated with an open file. With file layout definitions, you can read and write rowsets as easily as strings. If the file object isn't open or the definition doesn't exist, SetFileLayout fails.

You must execute SetFileLayout before you can use any other file layout methods or properties with a file object, otherwise those methods and properties return NULL or False.

Note. All provided PeopleTools records that have a prefix of PSFLD are related to file layout definitions.

Parameters

Parameter	***Description***
filelayoutname	Specify as a string the name of an existing file layout definition created in Application Designer.

Returns

A Boolean value: True if the method succeeds, False otherwise.

Example

The following example opens a data file, associates a file layout definition with it, reads and processes the first rowset from it, and closes the file.

```
&MYFILE.Open(&SOMENAME, "E");
If &MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT) then
   &SOMEROWSET = &MYFILE.ReadRowset();
   /* Process the contents of &SOMEROWSET */
Else
   /* Error - SetFileLayout failed */
End-If;
&MYFILE.Close();
```

See Also

Chapter 19, "File Class," CurrentRecord, page 1009; Chapter 19, "File Class," CreateRowset, page 984; Chapter 19, "File Class," ReadRowset, page 994; Chapter 19, "File Class," IgnoreInvalidId, page 1010 and Chapter 19, "File Class," WriteRowset, page 1006

Chapter 19, "File Class," File Layout Examples, page 1016

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Constructing File Layouts and Performing Data Interchanges"

SetPosition

Syntax

SetPosition*(position)*

Description

The SetPosition method sets the current read or write position in the external file associated with the file object executing this method. SetPosition works only with a file, which is opened in Update mode, and is designed to be used in combination with the GetPosition method to recover from interruptions during file access.

To start reading or writing from the beginning of a file, you must use SetPosition to set a read/write position of 0, immediately after you open the file in Update mode.

Parameters

Parameter	***Description***
position	The byte position in the file at which you want to continue reading or writing. This should either be 0 or a value you previously obtained with the GetPosition method and saved in a separate file or a database.

Warning! In Update mode, any write operation clears the file of all data that follows the position you set.

Note. Use SetPosition only for recovering from file access interruptions. Supplying your own value for SetPosition isn't recommended, except for position 0.

Returns

None.

Example

The following example reopens a file in Update mode, and sets the read position to the last saved checkpoint:

```
&MYFILE = GetFile(&SOMENAME, "U");
If &MYFILE.IsOpen Then
   /* Retrieve the value of the last saved checkpoint, &LASTPOS */
   &MYFILE.SetPosition(&LASTPOS);
   while &MYFILE.ReadLine(&SOMESTRING)
      /* Process the contents of each &SOMESTRING */
   End-While;
   &MYFILE.Close();
End-If;
```


Note. Currently, the effect of the Update mode and the GetPosition and SetPosition methods is not well defined for Unicode files. Use the Update mode only on files stored with a non-Unicode character set.

See Also

Chapter 19, "File Class," GetPosition, page 986 and Chapter 19, "File Class," Open, page 989

Chapter 19, "File Class," File Access Interruption Recovery, page 977

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

SetRecTerminator

Syntax

`SetRecTerminator`*`(Terminator)`*

Description

The SetRecTerminator method is a file layout method.

Use this method to specify the string of characters that indicate the end of a file record. Read operations use the value of SetRecTerminator to determine where each file record ends and the next one starts, and write operations append the value of SetRecTerminator to each record written.

This value defaults to the newline character appropriate to the platform where the file is being stored:

- a linefeed on UNIX systems
- a carriage return/linefeed combination on Windows systems

You need to specify a different record terminators only if you anticipate that part of the data in a file field that includes the newline character used on the storage platform; you must assign the record terminator a unique string that you know does not appear in a file field.

If you set the record terminator to Null ("") you eliminate linefeeds or carriage returns.

Parameters

Parameter	***Description***
Terminator	Specify the value you want used for the record terminator.

Returns

None.

Example

```
&File = GetFile(......);

if &File.IsOpen then
   /* set the terminator to be NULL */
   Local String &Term = "";
   &File.SetRecTerminator(&Term);
```

WriteLine

Syntax

`WriteLine`*`(string)`*

Description

The WriteLine method writes one string of text, *string,* to the output file associated with the file object executing this method, followed by a newline character appropriate to the platform where the file is being written. To build a single line using multiple strings, use the WriteString method.

Parameters

Parameter	***Description***
string	The string of text to be written.

Returns

None.

Example

The following example adds a line of text to an existing file:

```
&MYFILE.Open("somefile.txt", "A");
&MYFILE.WriteLine("This is the last line in the file.");
&MYFILE.Close();
```

The following example converts a file where the fields are separated with tabs into a file where the fields are separated with commas.


```
Local File &TABFILE, &CSVFILE;
Local string &FILE_NAME, &DATA, &NEWDATA;

&FILE_NAME = "Test.txt";
&TAB = Char(9);

&TABFILE = GetFile(&FILE_NAME, "r");
&FileName = &TABFILE.Name;
&POS = Find(".", &FileName);
&NEWFILE_NAME = Substring(&FileName, 1, &POS) | "dat";
&CSVFILE = GetFile(&NEWFILE_NAME, "N", %FilePath_Absolute);
If &TABFILE.IsOpen And
      &CSVFILE.IsOpen Then
   While &TABFILE.ReadLine(&DATA);
      &NEWDATA = Substitute(&DATA, &TAB, ",");
      &CSVFILE.WriteLine(&NEWDATA);
   End-While;
   &TABFILE.Close();
   &CSVFILE.Close();
End-If;
```

See Also

Chapter 19, "File Class," GetString, page 988; Chapter 19, "File Class," ReadLine, page 993; Chapter 19, "File Class," WriteString, page 1008 and Chapter 19, "File Class," WriteRaw, page 1003

WriteRaw

Syntax

WriteRaw*(RawBinary)*

Description

The WriteRaw method writes the contents of *RawBinary* to a file. This can be used for writing images, messages, or other types of raw binary data to a file.

Parameters

Parameter	***Description***
RawBinary	Specify the raw binary to be written to the file.

Returns

None.

Example

The following example writes employee photos (GIF files) from a record to a file.

```
Local File &FILE;
Local Record &REC;
Local SQL &SQL;

&REC = CreateRecord(Record.EMPL_PHOTO);
&SQL = CreateSQL("%SelectAll(:1)", Record.EMPL_PHOTO);

&FILE = GetFile("C:\temp\EMPL_PHOTO.GIF", "w", "a", %FilePath_Absolute);

While &SQL1.Fetch(&REC)
   &FILE.WriteRaw(&REC.EMPLOYEE_PHOTO.Value);
End-While;

&FILE.Close();
```

See Also

Chapter 19, "File Class," WriteLine, page 1002 and Chapter 19, "File Class," WriteString, page 1008

WriteRecord

Syntax

`WriteRecord`*`(record)`*

Description

The WriteRecord method is a file layout method. It writes the contents of the record object *record,* to the output file. (Remember, a record object contains only one row of data from an SQL table.)

You can use this method to build a transaction in the output file, one file record at a time, without having to instantiate and populate a rowset object. You can apply this method to any record whose structure and name matches that of a record defined in the current file layout.

Note. You must execute the SetFileId method from the file object before using WriteRecord, otherwise it returns False.

See Chapter 19, "File Class," SetFileLayout, page 998.

Note. When you're writing text to an XML file, special HTML characters, such as ampersands, lesser than or greater than symbols, and so on, are automatically converted to valid XML character string, such as &

Considerations Using XML With File Definition Tags

File Definition Tags have no effect when importing data. However, generally, during export, the File Definition Tag is added at the start and end of the data to create a valid XML file.

The one exception is when the file to which the data is being written during export has been opened in append mode. At that time, the file definition tag is not taken into consideration.

Considerations Using XML with File Definitions

If your file layout is defined as XML, WriteRecord doesn't add the closing tag for the record. You must write it yourself using the WriteLine method. This is because the code has no way of knowing when you want to write children records following the record just written out.

The following code shows an example of using WriteLine:

```
Local File &MYFILE;

&MYFILE = GetFile("XMLrecord.txt", "A");

If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.RECORDLAYOUT) Then
      &LN = CreateRecord(RECORD.QA_INVEST_LN);
      &SQL2 = CreateSQL("%Selectall(:1)", &LN);
      While &SQL2.Fetch(&LN)
         &MYFILE.WriteRecord(&LN);
         WriteLine("</QA_INVEST_LN>"); /* Add the closing tag */
      End-While;
   Else
   /* do error processing - filelayout not correct */
   End-If;
Else
/* do error processing - file not open */
End-If;

&MYFILE.Close();
```

Parameters

Parameter	***Description***
record	Specify the name of an existing record object to be written. You can use Rowset class methods such as GetField and built-in functions such as GetRecord to populate the record with data before writing it to the file.

Returns

A Boolean value: True if the method succeeds, False otherwise. This value is optional.

Example

The following example appends all the data in a record to an existing file:


```
Local File &MYFILE;

&MYFILE = GetFile("record.txt", "A");

If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.VOL_TEST) Then
      &LN = CreateRecord(RECORD.VOLNTER_ORG_TBL);
      &SQL2 = CreateSQL("%Selectall(:1)", &LN);
      While &SQL2.Fetch(&LN)
         &MYFILE.WriteRecord(&LN);
      End-While;
   Else
      /* do error processing - filelayout not correct */
   End-If;
Else
   /* do error processing -; file not open */
End-If;

&MYFILE.Close();
```

See Also

Chapter 19, "File Class," CreateRowset, page 984; Chapter 19, "File Class," ReadRowset, page 994; Chapter 19, "File Class," SetFileLayout, page 998 and Chapter 19, "File Class," WriteRowset, page 1006

Chapter 34, "Record Class," page 2077

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

WriteRowset

Syntax

`WriteRowset`*`(rowset`*` [, `*`Write_Data`*`])`

Description

The WriteRowset method is a file layout method. It writes the contents of a rowset object, *rowset,* to the output file associated with the file object executing this method. Regardless of whether the rowset contains just one or more than one transaction (level zero row), executing this method once writes the entire contents of the rowset to the output file.

Note. You must execute the SetFileLayout method from the file object before using WriteRowset, otherwise it returns False.

See Chapter 19, "File Class," SetFileLayout, page 998.

WriteRowset writes a rowset to a file only if the data in the component buffer has changed. You must specify the *WriteData* parameter as True if you want the WriteRowset method to force the rowset to be written to the file even if the buffer has not been changed.

Note. When you're writing text to an XML file, special HTML characters, such as ampersands, lesser than or greater than symbols, and so on, are automatically converted to valid XML character string, such as &amp.

Considerations Using Fixed Length Files With Numeric Fields

All numeric fields are right-justified in when writing to fixed length files. In addition, zeros are padded to the right after the decimal point if required.

Considerations Using XML With File Definition Tags

File Definition Tags have no effect when importing data. However, generally, during export, the File Definition Tag is added at the starting and end of the data to create a valid XML file.

The one exception is when the file to which the data is being written during export has been opened in append mode. At that time, the file definition tag is not taken into consideration.

Parameters

Parameter	***Description***
rowset	Specify the name of an existing rowset object that was instantiated with the File class CreateRowset or ReadRowset method. You can use Rowset class methods such as Select and built-in functions such as GetLevel0 to populate the rowset with data before writing it to the file.
WriteData	Specify whether to write the rowset data to the file whether or not the data in the buffer has changed. This parameter takes a Boolean value: True, write the data to the buffer regardless, False, only write the data if changed. The default value for this parameter is False.

Returns

A Boolean value: True if the method succeeds, False otherwise.

Example

```
Local File &MYFILE;
Local Rowset &FILEROWSET;

&MYFILE = GetFile("c:\temp\EMP_CKLS.txt", "A", %FilePath_Absolute);
If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.EMPL_CHECKLIST) Then
      &FILEROWSET = &MYFILE.CreateRowset();
      &FILEROWSET = GetLevel0();
      &MYFILE.WriteRowset(&FILEROWSET, True);
   Else
      /* file layout not found, do error processing */
   End-If;
Else
   /* file not opened, do error processing */
End-if;

&MYFILE.Close();
```

See Also

Chapter 19, "File Class," CreateRowset, page 984; Chapter 19, "File Class," ReadRowset, page 994; Chapter 19, "File Class," SetFileLayout, page 998; Chapter 19, "File Class," WriteRecord, page 1004 and Chapter 19, "File Class," ZeroExtend, page 1016

Chapter 34, "Record Class," page 2077

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

WriteString

Syntax

WriteString*(string)*

Description

The WriteString method writes one string of text to the output file associated with the file object executing this method, without any newline character. Each string written extends the current line in the file.

You can start a new line by using the WriteLine method to write the last part of the current line. WriteLine always adds a newline character appropriate to the platform where the file is being written, whether you supply a character string of any length, or a null string.

Parameters

Parameter	*Description*
string	A string variable containing the text to be written.

Returns

None.

Example

The following example opens an empty file, writes two lines of text to it without a final newline, and closes it:

```
&MYFILE.Open("somefile.txt", "W");
&MYFILE.WriteString("This is the first ");
&MYFILE.WriteString("line in the file.");
&MYFILE.WriteLine("");
&MYFILE.WriteString("This second line is not terminated.");
&MYFILE.Close();
```

See Also

Chapter 19, "File Class," ReadLine, page 993; Chapter 19, "File Class," WriteLine, page 1002 and Chapter 19, "File Class," WriteRaw, page 1003

File Class Properties

In this section, we discuss each File Class property.

CurrentRecord

Description

This property is a file layout property. This property returns the current record as a string. The CurrentRecord property is used in combination with the SetFileId and ReadRowset methods and IsNewFileId property when reading files that contain data based on multiple file layouts.

If the ReadRowset method returns NULL, either the current record is a FileId record or the end of file has been reached. The IsNewFileId enables you to determine which. If it is a FileId, you can parse the string returned by CurrentRecord to determine the file layout definition to be used.

Note. SetFileId, CurrentRecord, and IsNewFileId don't apply to CSV and XML format input files. You can implement multiple file layouts only with fixed format files.

This property is read-only.

Example

```
&RECSTRING = &MYFILE.CurrentRecord;
```

See Also

Chapter 19, "File Class," SetFileId, page 996; Chapter 19, "File Class," ReadRowset, page 994 and Chapter 19, "File Class," IsNewFileId, page 1011

Chapter 19, "File Class," Multiple File Layouts, page 1029

IgnoreInvalidId

Description

This property is a file layout property that's used in combination with the ReadRowset method. It returns a Boolean value that specifies whether file records with invalid FileIds are ignored. Each time ReadRowset is executed, it may encounter a file record that doesn't qualify as part of the rowset because its File ID isn't part of the current file layout, or because it occurs in an invalid position in the rowset. If IgnoreInvalidId is False, the PeopleCode program terminates; if IgnoreInvalidId is True, ReadRowset ignores the invalid file record. The default value is True.

This property is read-write.

Example

```
&MYFILE.IgnoreInvalidId = False;
```

See Also

Chapter 19, "File Class," ReadRowset, page 994

IsError

Description

This property is a file layout property. It returns a Boolean value indicating whether a field error condition was generated by the last file layout method executed. If an error condition was generated, IsError returns True; if not, IsError returns False. The default value is False.

This property is read-only.

Example

The following example shows where IsError would be used:

```
&MYFILE.Open(&SOMENAME, "R");
&MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT);
&MYROWSET = &MYFILE.ReadRowset();
If &MYFILE.IsError Then
   /* Examine the EditError property of each field in the rowset
      to find the one with the error, and respond accordingly */
End-If;
&MYFILE.Close();
```

See Also

Chapter 19, "File Class," Multiple File Layouts, page 1029

IsNewFileId

Description

This property is a file layout property. It returns a Boolean value indicating whether a FileId file record has been encountered. When the ReadRowset method reads a transaction from an input file, it examines the current file record following the transaction. If that file record is a FileId file record, IsNewFileId returns True; if not, IsNewFileId returns False.

IsNewFileId is used in combination with the SetFileId method and CurrentRecord property when reading files that require multiple file layouts.

Note. SetFileId, CurrentRecord, and IsNewFileId don't apply to CSV and XML format input files. You can use only fixed format files to implement multiple file layouts.

This property is read-only.

Example

```
&MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT) then  /* Set the first layout */
&MYFILE.SetFileId("999", 1);  /* Set the FileId */
&SOMEROWSET = &MYFILE.ReadRowset();  /* Read the first rowset */
While &SOMEROWSET <> NULL
   If &MYFILE.IsNewFileId Then
      /* Examine &MYFILE.CurrentRecord for the next layout */
      /* Set the next layout */
      &MYFILE.SetFileId("999", 1);  /* Set the FileId */
   End-If;
   /* Process the current &SOMEROWSET */
   &SOMEROWSET = &MYFILE.ReadRowset();  /* Read the next rowset */
End-While;
```


See Also

Chapter 19, "File Class," ReadRowset, page 994; Chapter 19, "File Class," SetFileId, page 996 and Chapter 19, "File Class," CurrentRecord, page 1009

IsOpen

Description

This property returns a Boolean value indicating whether the file is open. If the file object is open, IsOpen returns True; if not, IsOpen returns False.

This property is read-only.

Example

The following example opens a file, writes a line to it, and closes it:

```
&MYFILE.Open("item.txt", "W");
If &MYFILE.IsOpen Then
   &MYFILE.WriteLine("Some text.");
   &MYFILE.Close();
End-If;
```

Name

Description

This property returns as a string the name of the external file. If no file is currently associated with the file object, Name returns a NULL string.

This property is read-only.

Example

```
&TMP = &MYFILE.Name;
```

TerminateLines

Description

Use the TerminateLines property to indicate whether all output data lines are to be terminated at the end of the defined data length. This property is a file layout property associated with fixed file layouts only.

This property takes a Boolean value: True to force output data lines to be terminated at the end of the defined data length; False to keep all lines all at the maximum of their own length or that of their children. The default value is false.

This property is read-write.

Example

```
If &FILE_CREATED = "N" Then
   &FILE_CREATED = "Y";
   &FILENAME = &MSGNAME | "_" | &PROCESS_INSTANCE | "_*.out";
   &FILE = GetFile(&FILENAME, "N", "U");
   &FILE.SetFileLayout(@("FILELAYOUT." | &MSGNAME));
   &FILE.TerminateLines = True;
   &RS_LVL0 = &FILE.CreateRowset();
   &REC_MSG_LVL0 = &RS_LVL0(&ROWCNT0).GetRecord(1);

   If EO_BATLIB_AET.CREATE_FILE_FLG = "C" Then
      &STRING = "998       " | &MSGNAME;
      &FILE.WriteLine(&STRING);
   End-If;
End-If;
```

UseSpaceForNull

Description

Use the UseSpaceForNull property to specify whether the WriteRowset method writes <qualifier><qualifier> (<qualifier>-space-<qualifier>) for all the Null or empty character fields of a CSV-type file layout, or if the method writes <qualifier><qualifier> (<qualifier>-<qualifier>).

This property takes a Boolean value: false if WriteRowset writes <qualifier>-<qualifier>, true if WriteRowset writes <qualifier>-space-<qualifier>. The default is False.

Note. The state of this property has no effect on the behavior of the ReadRowset method. It also has no effect when the associated file layout is of type fixed or XML.

This property is read-write.

Example

```
Local File &LOGFILE;
Local File &FILE1;
Local File &FILE2;
Local Record &REC1;
Local SQL &SQL1;
Local Rowset &RS1;
Local Row &ROW1;

&LOGFILE = GetFile("C:\Temp\currency.log", "W", "A", %FilePath_Absolute);
If (&LOGFILE = Null) Then
   Exit;
End-If;
If Not (&LOGFILE.SetFileLayout(FileLayout.CURRENCY_FL)) Then
   &LOGFILE.WriteLine("SetFileLayout() on LOGFILE failed.");
End-If;

&FILE1 = GetFile("C:\Temp\currency_nonspaced.csv", "W", "A", %FilePath_Absolute);
If (&FILE1 = Null) Then
   &LOGFILE.WriteLine("FATAL ERROR:  GetFile() on non-spaced output file failed.");
   &LOGFILE.Close();
   Exit;
End-If;
If Not (&FILE1.SetFileLayout(FileLayout.CURRENCY_FL)) Then
   &LOGFILE.WriteLine("FATAL ERROR:  SetFileLayout() on non-spaced output file⇒
 failed.");
   &LOGFILE.Close();
   &FILE1.Close();
   Exit;
End-If;

&FILE2 = GetFile("C:\Temp\currency_spaced.csv", "W", "A", %FilePath_Absolute);
If (&FILE2 = Null) Then
   &LOGFILE.WriteLine("FATAL ERROR:  GetFile() on spaced output file failed.");
   &LOGFILE.Close();
   &FILE1.Close();
   Exit;
End-If;
If Not (&FILE2.SetFileLayout(FileLayout.CURRENCY_FL)) Then
   &LOGFILE.WriteLine("FATAL ERROR:  SetFileLayout() on spaced output file⇒
 failed.");
   &LOGFILE.Close();
   &FILE1.Close();
   &FILE2.Close();
   Exit;
End-If;

&REC1 = CreateRecord(Record.CURRENCY_CD_TBL);
If (&REC1 = Null) Then
   &LOGFILE.WriteLine("FATAL ERROR:  CreateRecord() on record failed.");
   &LOGFILE.Close();
   &FILE1.Close();
   &FILE2.Close();
   Exit;
End-If;

&RS1 = CreateRowset(Record.CURRENCY_CD_TBL);
If (&RS1 = Null) Then
   &LOGFILE.WriteLine("FATAL ERROR:  CreateRowset() on record failed.");
   &LOGFILE.Close();
   &FILE1.Close();
```

```
      &FILE2.Close();
      Exit;
   End-If;

   &SQL1 = CreateSQL("%Selectall(:1)", &REC1);
   If (&SQL1 = Null) Then
      &LOGFILE.WriteLine("FATAL ERROR:  CreateSQL() failed.");
      &LOGFILE.Close();
      &FILE1.Close();
      &FILE2.Close();
      Exit;
   End-If;

   If (&FILE1.UseSpaceForNull) Then
      &LOGFILE.WriteLine("UseSpaceForNull is True on non-spaced output, by default.");
   Else
      &LOGFILE.WriteLine("UseSpaceForNull is False on non-spaced output, by⇒
 default.");
   End-If;
   If (&FILE2.UseSpaceForNull) Then
      &LOGFILE.WriteLine("UseSpaceForNull is True on spaced output, by default.");
   Else
      &LOGFILE.WriteLine("UseSpaceForNull is False on spaced output, by default.");
   End-If;

   &FILE1.UseSpaceForNull = True;
   If (&FILE1.UseSpaceForNull) Then
      &LOGFILE.WriteLine("Setting UseSpaceForNull to True succeeded on non-spaced⇒
 output.");
   Else
      &LOGFILE.WriteLine("Setting UseSpaceForNull to True failed on non-spaced⇒
 output.");
   End-If;
   &FILE2.UseSpaceForNull = True;
   If (&FILE2.UseSpaceForNull) Then
      &LOGFILE.WriteLine("Setting UseSpaceForNull to True succeeded on spaced⇒
 output.");
   Else
      &LOGFILE.WriteLine("Setting UseSpaceForNull to True failed on spaced output.");
   End-If;

   &FILE1.UseSpaceForNull = False;
   If (&FILE1.UseSpaceForNull) Then
      &LOGFILE.WriteLine("Setting UseSpaceForNull to False failed on non-spaced⇒
 output.");
   Else
      &LOGFILE.WriteLine("Setting UseSpaceForNull to False succeeded on non-spaced⇒
 output.");
   End-If;
   &FILE2.UseSpaceForNull = False;
   If (&FILE2.UseSpaceForNull) Then
      &LOGFILE.WriteLine("Setting UseSpaceForNull to False failed on spaced output.");
   Else
      &LOGFILE.WriteLine("Setting UseSpaceForNull to False succeeded on spaced⇒
 output.");
   End-If;

   &FILE1.UseSpaceForNull = False;
   &FILE2.UseSpaceForNull = True;
   &I = 1;
   While &SQL1.Fetch(&REC1)
      &ROW1 = &RS1.GetRow(1);
```



```
   &REC1.CopyFieldsTo(&ROW1.CURRENCY_CD_TBL);
   &FILE1.WriteRowset(&RS1, True);
   &FILE2.WriteRowset(&RS1, True);
   REM   &LOGFILE.WriteLine("Got row " | String(&I));
   &I = &I + 1;
End-While;

&FILE1.Close();
&FILE2.Close();
&LOGFILE.Close();
```

See Also

Chapter 19, "File Class," WriteRowset, page 1006

ZeroExtend

Description

Use this property to specify whether or not the WriteRowset method zero-extends the value it writes for decimal fields for CSV or XML format files.

For example, for a decimal field defined as 6.3, the value 1.12 will be written as follows depending on the value of ZeroExtend:

True: 1.120

False: 1.12

This property takes a Boolean value: true if WriteRowset zero-extends the value it writes; false otherwise. The default value is true.

Note. This property has no effect in the case of a fixed-position format file. In addition, it does not affect the behavior of the ReadRowset method.

This property is read-write.

See Also

Chapter 19, "File Class," WriteRowset, page 1006

File Layout Examples

If your data is hierarchical in nature, or based on existing PeopleSoft records or pages, you want to use a File Layout definition for reading and writing your data, rather than doing it line by line (or field by field.)

For example, suppose you wanted to write all the information from a record to a file. You can use the WriteRecord method to write all the data from the record, instead of having to loop through every field, find the value, and write it to the file.

In addition, you could write all the information from a transaction (or several transactions) from a page to a file. Each transaction can be considered a *rowset.* A rowset can contain more than one record and is generally composed in a hierarchical structure. You could create a File Layout definition that has the same structure as the page (or component), and use the WriteRowset method. If you have a file that contains data in the correct format, you can use the ReadRowset method to read the data from the file to the page.

Each file layout is associated with a *format.* This format specifies the type of data in the files. You specify the format as part of the File Layout Properties. You can only specify one format for a file layout. Available formats are:

- FIXED (default)
- CSV
- XML

The file layout methods and properties use this information to handle each file type in a transparent manner. Generally, you don't need to do anything different based on file type. Any exceptions are noted in the documentation.

Note. Unlike other PeopleTools definitions, records and field are *copied* to a File Layout definition. There are no pointers. This means if you change a record definition (add or remove a field) you must change the File Layout definition also. The changes are not automatically propagated. This is why the documentation refers to these elements as file records, file fields, and so on, to show that they are no longer part of the original definition they were created from.

PeopleSoft recommends regenerating all file layout definitions after any upgrade, to avoid any corruption caused by changes to the database.

See Using Standalone Rowsets for more examples of writing from and reading to files using File Layout and standalone rowsets.

See Also

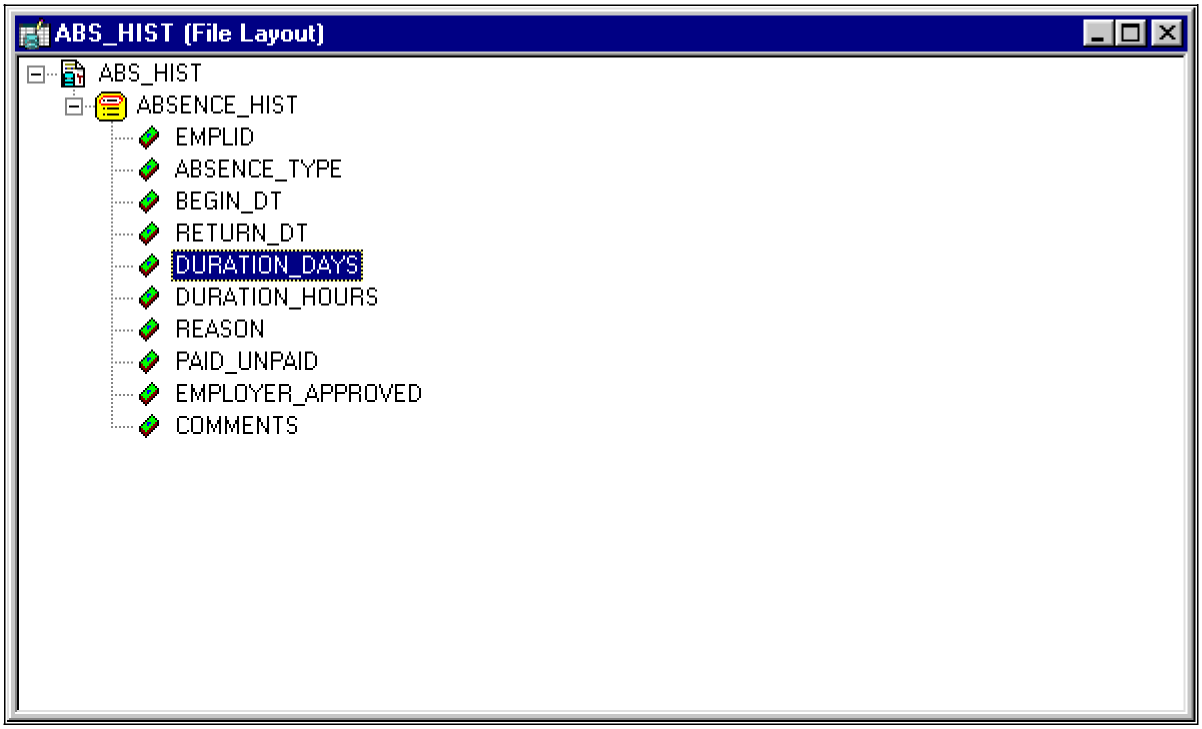
PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Constructing File Layouts and Performing Data Interchanges"

WriteRecord Example

In the following example, the File Layout definition is based on the record ABSENCE_HISTORY, and looks like this:



Example File Layout definition (ABS_HIST)

You should note the following about the using the WriteRecord method:

- Not all the fields in the File Layout definition and the record have to match. The WriteRecord method, like all File Layout methods, applies only to the *like-named* fields. If there are additional fields in the record or in the File Layout definition, they are ignored.
- The WriteRecord method writes only to *like-named* records. If you rename a record after you use it to create a File Layout definition, you must rename it to the exact same name in your File Layout. Because WriteRecord writes like-named records, the same file layout definition can contain more than one record.
- The WriteRecord method takes a *record object* as its parameter. A populated record object references a *single row* of data in the SQL table. This is why a SQL Fetch statement is used in a condition around the WriteRecord method. This fetches every row of data from the SQL table, then writes it to the file.

The following code writes the ABSENCE_HIST record to the file record.txt.

```
Local Record &RecLine;
Local File &MYFILE;
Local SQL &SQL2;

&MYFILE = GetFile("record.txt", "A");

If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FileLayout.ABS_HIST) Then
      &RecLine = CreateRecord(RECORD.ABSENCE_HIST);
      &SQL2 = CreateSQL("%Selectall(:1)", &RecLine);
      While &SQL2.Fetch(&RecLine)
         &MYFILE.WriteRecord(&RecLine);
      End-While;
   Else
      /* do error processing -; filelayout not correct */
   End-If;
Else
   /* do error processing -; file not open */
End-If;

&MYFILE.Close();
```

If you wanted to write only changed records to the file, you could add the following code that is set in bold font:

```
While &SQL2.Fetch(&RecLine)
   If &RecLine.IsChanged Then
         &MYFILE.WriteRecord(&RecLine);
   End-If;
End-While;
```

The following is the first part of a sample data file created by the previous code. The field format is FIXED:

```
8001       VAC 09/12/1981 09/26/1981 14  0                                P Y
8001       VAC 03/02/1983 03/07/1983 5   0                                P Y
8001       VAC 08/26/1983 09/10/1983 13  0                                P Y
8105       CNF 02/02/1995 ??/??/     0   0                                U N
8516       MAT 06/06/1986 08/01/1986 56  0                                P Y
8516       SCK 08/06/1988 08/07/1988 1   0                                P Y
8516       VAC 07/14/1987 07/28/1987 14  0                                P Y
8553       JUR 12/12/1990 12/17/1990 5   0   Local Jury Duty              P N
8553       MAT 02/20/1992 10/01/1992 224 0   Maternity Leave              U N
8553       MAT 08/19/1994 03/01/1995 194 0   Maternity-2nd child          U Y
8553       PER 04/15/1993 04/19/1993 4   0                                U N⇒
 Personal Day required
8553       SCK 01/28/1987 01/30/1987 2   0   Hong Kong Flu                P N
8553       SCK 08/02/1988 08/03/1988 1   0   Sick                         P N
8553       SCK 09/12/1995 09/13/1995 1   0                                P N
8641       VAC 06/01/1988 06/15/1988 14  0                                P Y
8641       VAC 07/01/1989 07/15/1989 14  0                                P Y
G001       MAT 07/02/1991 09/28/1991 88  0   3-month Maternity leave      P Y⇒
 Maternity will be paid as 80% of Claudia's current salary.
```

If a record in the File Layout definition has a File Record ID, each line in the file referencing this record is prefaced with this number. The File Record ID is not a field in the data itself. It is also referred to as the *rowid.* File Record IDs can be useful when the file you're producing refers to more than one record.

File Record IDs can be used only with File Layout definitions that have a type of FIXED. If a File Layout definition has only one level then File Record IDs can be omitted. But for File Layout definitions with more than one level, you *must* use File Record IDs.

The following is sample file, produced with the same code, but with a File Record ID of 101 added:


```
101 8001       VAC 09/12/1981 09/26/1981 14  0                              P Y
101 8001       VAC 03/02/1983 03/07/1983 5   0                              P Y
101 8001       VAC 08/26/1983 09/10/1983 13  0                              P Y
101 8105       CNF 02/02/1995 ??/??/     0   0                              U N
101 8516       MAT 06/06/1986 08/01/1986 56  0                              P Y
101 8516       SCK 08/06/1988 08/07/1988 1   0                              P Y
101 8516       VAC 07/14/1987 07/28/1987 14  0                              P Y
101 8553       JUR 12/12/1990 12/17/1990 5   0   Local Jury Duty            P N
101 8553       MAT 02/20/1992 10/01/1992 224 0   Maternity Leave            U N
101 8553       MAT 08/19/1994 03/01/1995 194 0   Maternity-2nd child        U Y
101 8553       PER 04/15/1993 04/19/1993 4   0                              U N⇒
 Personal Day required
101 8553       SCK 01/28/1987 01/30/1987 2   0   Hong Kong Flu              P N
101 8553       SCK 08/02/1988 08/03/1988 1   0   Sick                       P N
101 8553       SCK 09/12/1995 09/13/1995 1   0                              P N
101 8641       VAC 06/01/1988 06/15/1988 14  0                              P Y
101 8641       VAC 07/01/1989 07/15/1989 14  0                              P Y
101 G001       MAT 07/02/1991 09/28/1991 88  0   3-month Maternity leave    P Y⇒
 Maternity will be paid as 80% of Claudia's current salary.
```

Note. File positions start at 1, not 0. When you specify a File Record ID, make sure that the starting position of the Record ID is greater than 0.

ReadRecord Example

This following example uses the same File Layout definition as the previous example. The file format is CSV. The fields are separated by commas and delimited by double-quotes.

```
"8001","VAC","09/12/1981","09/26/1981","14","0","","P","Y",""
"8001","VAC","03/02/1983","03/07/1983","5","0","","P","Y",""
"8001","VAC","08/26/1983","09/10/1983","13","0","","P","Y",""
"8105","CNF","02/02/1995","??/??/","0","0","","U","N",""
"8516","MAT","06/06/1986","08/01/1986","56","0","","P","Y",""
"8516","SCK","08/06/1988","08/07/1988","1","0","","P","Y",""
"8516","VAC","07/14/1987","07/28/1987","14","0","","P","Y",""
"8553","JUR","12/12/1990","12/17/1990","5","0","Local Jury Duty","P","N",""
"8553","MAT","02/20/1992","10/01/1992","224","0","Maternity Leave","U","N",""
"8553","MAT","08/19/1994","03/01/1995","194","0","Maternity-2nd child","U","Y",""
"8553","PER","04/15/1993","04/19/1993","4","0","","U","N","Personal Day required"
"8553","SCK","01/28/1987","01/30/1987","2","0","Hong Kong Flu","P","N",""
"8553","SCK","08/02/1988","08/03/1988","1","0","Sick","P","N",""
"8553","SCK","09/12/1995","09/13/1995","1","0","","P","N",""
"8641","VAC","06/01/1988","06/15/1988","14","0","","P","Y",""
"8641","VAC","07/01/1989","07/15/1989","14","0","","P","Y",""
"G001","MAT","07/02/1991","09/28/1991","88","0","3-month Maternity leave","P","Y",⇒
"Maternity will be paid as 80% of Claudia's current salary."
```

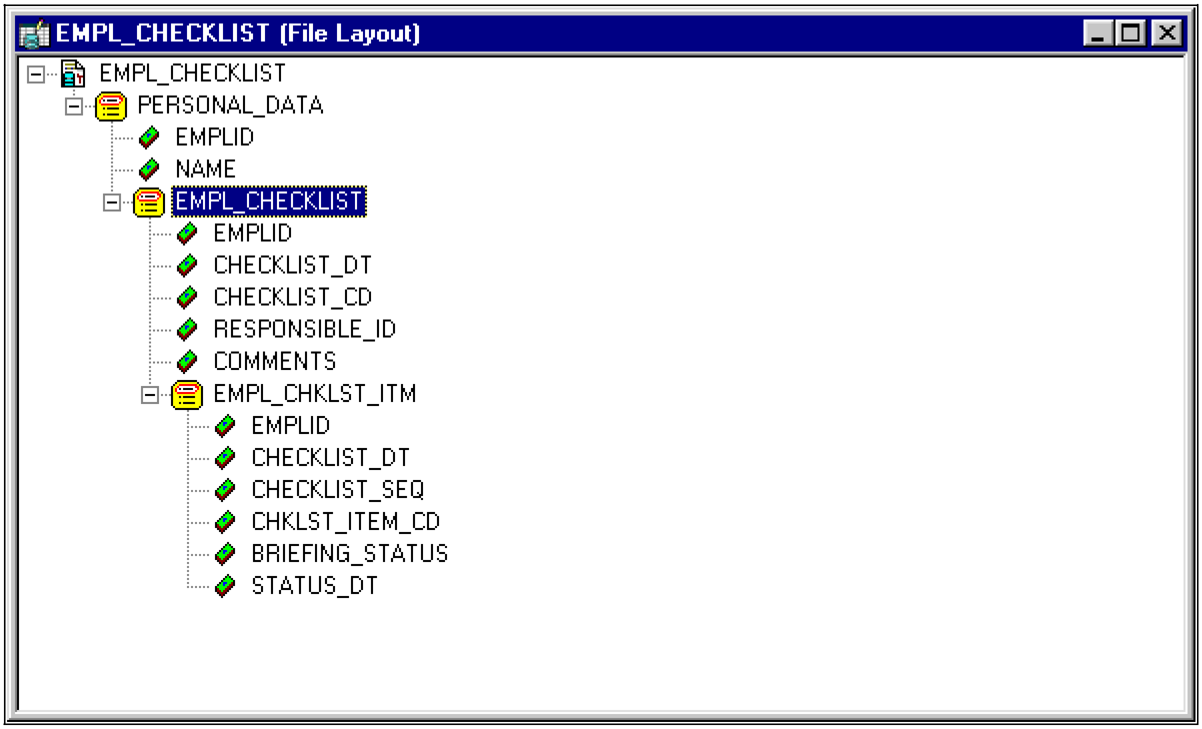
To read in the previous CSV file we use the following PeopleCode. It reads the file into a temporary record. First each line of the file is read into a string. The string is split into an array, with the value of each field in the array becoming an element in the array. The value of each field in the record is assigned a value from the array. After additional processing (for example, converting strings into dates or numbers, verifying data, and so on) the record can be inserted into the database. To insert the final data into the database, this code must be associated with a PeopleCode event that allows database updates, that is, SavePreChange, WorkFlow, SavePostChange, and so on. This code could also be used as part of an Application Engine program.


```
Local File &MYFILE;
Local Record &REC;
Local array of string &ARRAY;

&MYFILE = GetFile("c:\temp\vendor.txt", "R", %FilePath_Absolute);
&REC = CreateRecord(RECORD.ABS_HIST_TEST);
&ARRAY = CreateArrayRept("", 0);

If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.ABS_HIST) Then
      While &MYFILE.ReadLine(&STRING);
         &ARRAY = Split(&STRING, ",");
         For &I = 1 To &REC.FieldCount
            &REC.GetField(&I).Value = &ARRAY[&I];
         End-For;
      /* do additional processing here for converting values */
         &REC.Insert();
      End-While;
   Else
      /* do error processing - filelayout not correct */
   End-If;
Else
   /* do error processing - file not open */
End-If;

&MYFILE.Close();
```

Note. You can't read a file that contains a thousands separator for numeric fields. You must strip out the separator before you try to read in the file.

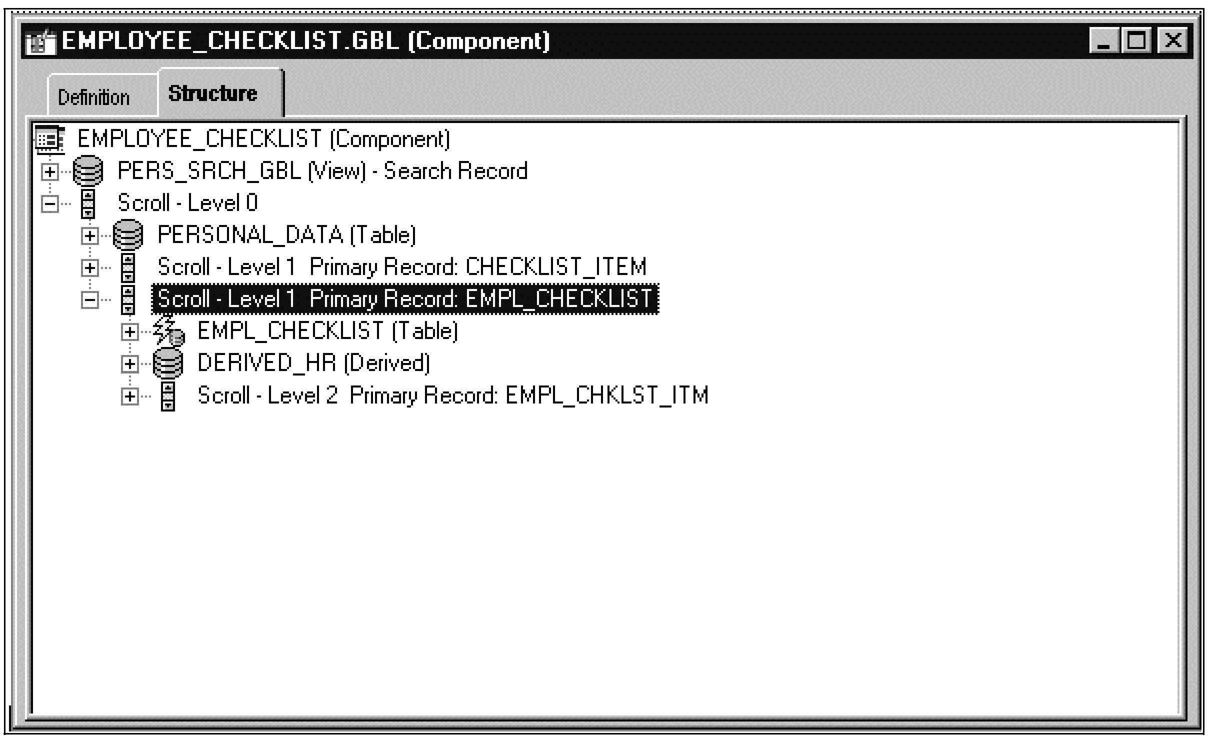
WriteRowset Example

In the following example, the File Layout definition is based on the component EMPL_CHECKLIST, and looks like this:



Example File Layout definition (EMPL_CHECKLIST)

Here's the structure of the component EMPLOYEE_CHECKLIST:



EMPLOYEE_CHECKLIST Component structure

Note that:

- Every *field* in the two structures don't have to match (that is, every field or record that's in the file layout doesn't have to be in the component, and vice versa.)
- The two *structures* must be the same. That is, if the component has PERSONAL_DATA at level zero, and EMPL_CHECKLIST at level one, the file layout must have the same hierarchy.

The following example uses the previous File Layout definition to copy data from the EMPL_CHECKLIST page into a file.

The CreateRowset function creates an empty rowset that has the structure of the file layout definition. The GetRowset function is used to get all the data from the component and copy it into the rowset. The GetLevel0 function copies all *like-named* fields to *like-named* records. The WriteRowset method writes all the component data to the file. Because this code runs on the server, an absolute file path is used.

See *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetRowset.

Example

The following is the PeopleCode for this example.

```
Local File &MYFILE;
Local Rowset &FILEROWSET;

&MYFILE = GetFile("c:\temp\EMP_CKLS.txt", "A", %FilePath_Absolute);
If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.EMPL_CHECKLIST) Then
      &FILEROWSET = &MYFILE.CreateRowset();
      &FILEROWSET = GetLevel0();
      &MYFILE.WriteRowset(&FILEROWSET, True);
   Else
      /* file layout not found, do error processing */
   End-If;
Else
   /* file not opened, do error processing */
End-if;

&MYFILE.Close();
```

The following is a sample data file created by the previous code:


```
8113       Frumman,Wolfgang
           08/06/1999 000001          8219       Going to London office
                     100     000015 I 08/06/1999
                     200     000030 I 08/06/1999
                     300     000009 I 08/06/1999
                     400     000001 I 08/06/1999
                     500     000011 I 08/06/1999
                     600     000002 I 08/06/1999
                     700     000021 I 08/06/1999
                     800     000024 I 08/06/1999
                     900     000004 I 08/06/1999
                     1000    000006 I 08/06/1999
           09/06/1999 000004          7707       What to do after he arrives
                     100     000022 I 08/06/1999
                     200     000008 I 08/06/1999
                     300     000018 I 08/06/1999
                     400     000019 I 08/06/1999
8101       Penrose,Steven
           07/06/1999 000006          8229       New hire
                     1       000033 I 08/06/1999
                     2       000034 I 08/06/1999
                     3       000035 I 08/06/1999
                     4       000036 I 08/06/1999
                     5       000037 I 08/06/1999
                     6       000038 I 08/06/1999
                     7       000039 I 08/06/1999
                     8       000040 I 08/06/1999
                     9       000041 I 08/06/1999
                     10      000042 I 08/06/1999
```

When you create the File Layout definition, you can choose for each field whether to inherit a value from a higher level record field. If a value is inherited, it is written only *once* to the file, the first time it's encountered.

In the previous data example, there are three records: PERSONAL_DATA, EMPL_CHECKLIST, and EMPL_CHKLST_ITM. The field EMPLID is on all three records, but is written to the file only the first time a new value is encountered. So, the value for EMPLID is inherited by EMPL_CHECKLIST, and the value for EMPL_CHKLST_ITM is inherited from PERSONAL_DATA. The field CHECKLIST_DT is on EMPL_CHECKLIST and EMPL_CHKLST_ITM. However, the field value appears only once, in the parent record, and not in the child record.

If a record in the File Layout definition has a File Record ID, each line in the file referencing this record will be prefaced with this number. The File Record ID is not a field in the data itself. It is also referred to as the *rowid.* File Record IDs can be useful when the file being created refers to more than one record. File Record IDs can be used only with File Layout definitions that have a type of FIXED.

The following is sample file, produced with the same code, but with a File Record IDs added to all the records. 001 was added to the first level, 002 to the second, and 003 to the third.


```
001 8113       Frumman,Wolfgang
002            08/06/1999      000001 8219        Going to London office
003                       100    000015 I 10/13/1999
003                       200    000030 I 10/13/1999
003                       300    000009 I 10/13/1999
003                       400    000001 I 10/13/1999
003                       500    000011 I 10/13/1999
003                       600    000002 I 10/13/1999
003                       700    000021 I 10/13/1999
003                       800    000024 I 10/13/1999
003                       900    000004 I 10/13/1999
003                       1000   000006 I 10/13/1999
002            09/06/1999      000004 7707        What to do after he arrives
003                       100    000022 I 10/13/1999
003                       200    000008 I 10/13/1999
003                       300    000018 I 10/13/1999
003                       400    000019 I 10/13/1999
001 8101       Penrose,Steven
002            10/13/1999      000006 8229        New hire
003                       1      000033 I 10/13/1999
003                       2      000034 I 10/13/1999
003                       3      000035 I 10/13/1999
003                       4      000036 I 10/13/1999
003                       5      000037 I 10/13/1999
003                       6      000038 I 10/13/1999
003                       7      000039 I 10/13/1999
003                       8      000040 I 10/13/1999
003                       9      000041 I 10/13/1999
003                       10     000042 I 10/13/1999
```

ReadRowset Example

The following program reads all the rowsets from a file and updates a work scroll with that data. The work scroll isn't hidden on the page: it's created in the Component buffer from existing records using CreateRowset. The structure of the work scroll and the file layout are identical: that is, they're composed of two records, and the names of the records in the file layout are exactly the same as the names of the record definitions.


```
Local File &CHARTINPUT_F;
Local Rowset &INPUT_ROWSET, &TEMP_RS, &WORK_DATA;
Local Record &WRK_DATA;

&filename = "c:\temp\test.txt";
If FileExists(&filename, %FilePath_Absolute) Then
   &CHARTINPUT_F = GetFile(&filename, "R", "A", %FilePath_Absolute);
Else
   Exit;
End-If;

&CHARTINPUT_F.SetFileLayout(FileLayout.CHART_INFO);

/* Create rowset to be read into
NOTE that you have to start at LOWEST level of rowset */

&TEMP_RS = CreateRowset(RECORD.CHART_ITEM);
&WORK_DATA = CreateRowset(RECORD.CHART_DATA, &TEMP_RS);
&INPUT_ROWSET = CreateRowset(RECORD.CHART, &WORK_DATA);

While &INPUT_ROWSET <> Null
   &INPUT_ROWSET = &CHARTINPUT_F.ReadRowset();
   &INPUT_ROWSET.CopyTo(&WORK_DATA);
   /* do processing -- Though file may contain more than one level zero
   Component processor only allows one level zero at a time */
End-While;
```

File Rowset Considerations

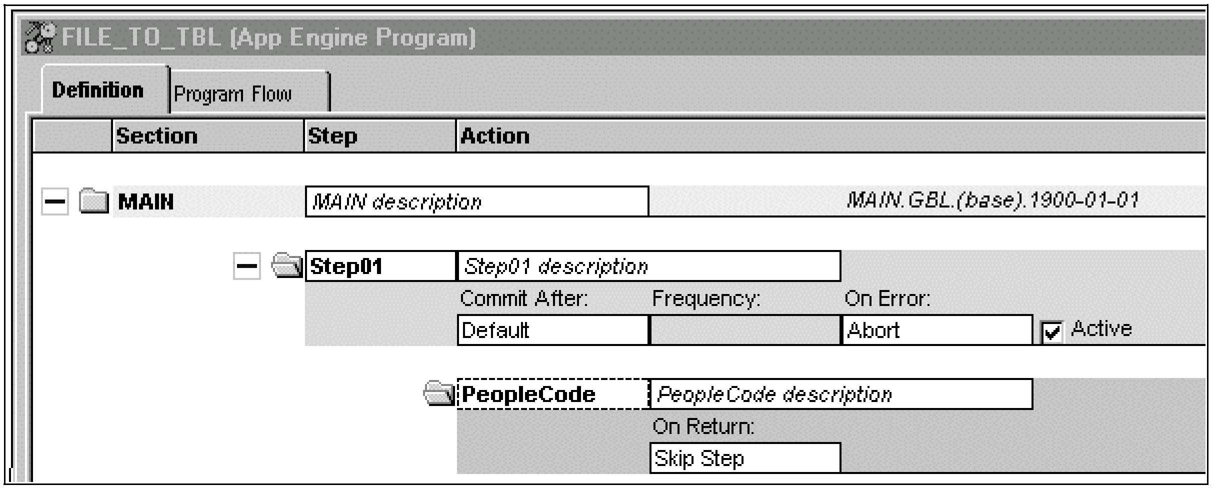
The following are considerations for when you use rowsets with files.

- Although you can create a File Layout definition with more than four levels of hierarchy, a rowset created from Component buffer data can contain only four levels (level zero through 3). Any additional levels of data are ignored.
- ReadRowset populates the rowset with one transaction from the file. (A transaction is considered to be one instance of level zero data contained in a file record, plus all of its subordinate data.) WriteRowset writes one transaction to a file.

Application Engine Example

You can also use PeopleCode in an Application Engine program to either write to or read from files. This example isn't a proper Application Engine program: it contains the PeopleCode only for opening and reading from a file. However, it's included here as starting point for your own Application Engine programs.

Here is the Application Engine program:



Application Engine example program

Here is the PeopleCode in the step 1.

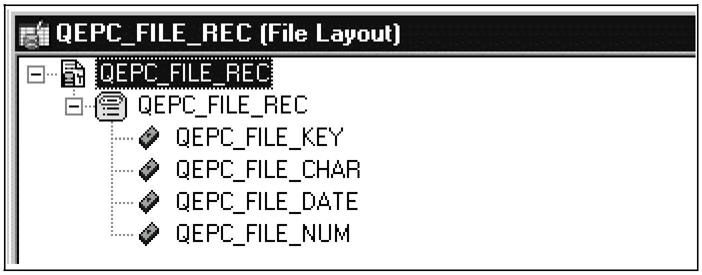
```
Local File &FILE;
Local Record &REC;
Local Rowset &FRS;

&FILE = GetFile("TEST.txt", "R");
&REC = CreateRecord(Record.QEPC_FILE_REC);
&SQL = CreateSQL("%Insert(:1)");

If Not &FILE.IsOpen Then
   Error ("TEST: failed file open");
Else
   If Not &FILE.SetFileLayout(FileLayout.QEPC_FILE_REC) Then
      Error ("TEST: failed SetFilelayout");
   Else
      &FRS = &FILE.ReadRowset();
      While &FRS <> Null
         &FRS.GetRow(1).QEPC_FILE_REC.CopyFieldsTo(&REC);
         &SQL.execute(&REC);
         &FRS = &FILE.ReadRowset();
      End-While;
   End-If;
   &FILE.Close();
End-If;
```

The example Application Engine program reads the following CSV file:

```
"TEST2","1ST","01/01/1901",10
"TEST2","2ND","01/01/1902",20
"TEST2","3RD","01/01/1903",30
"TEST2","4TH","01/01/1904",40
```

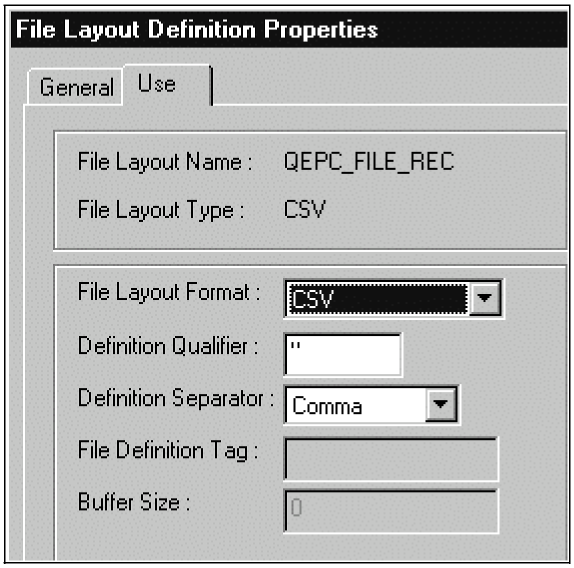
Note that the last field has no qualifier.

The File Layout used to read this record has the following form:



QEPC_FILE_REC File Layout

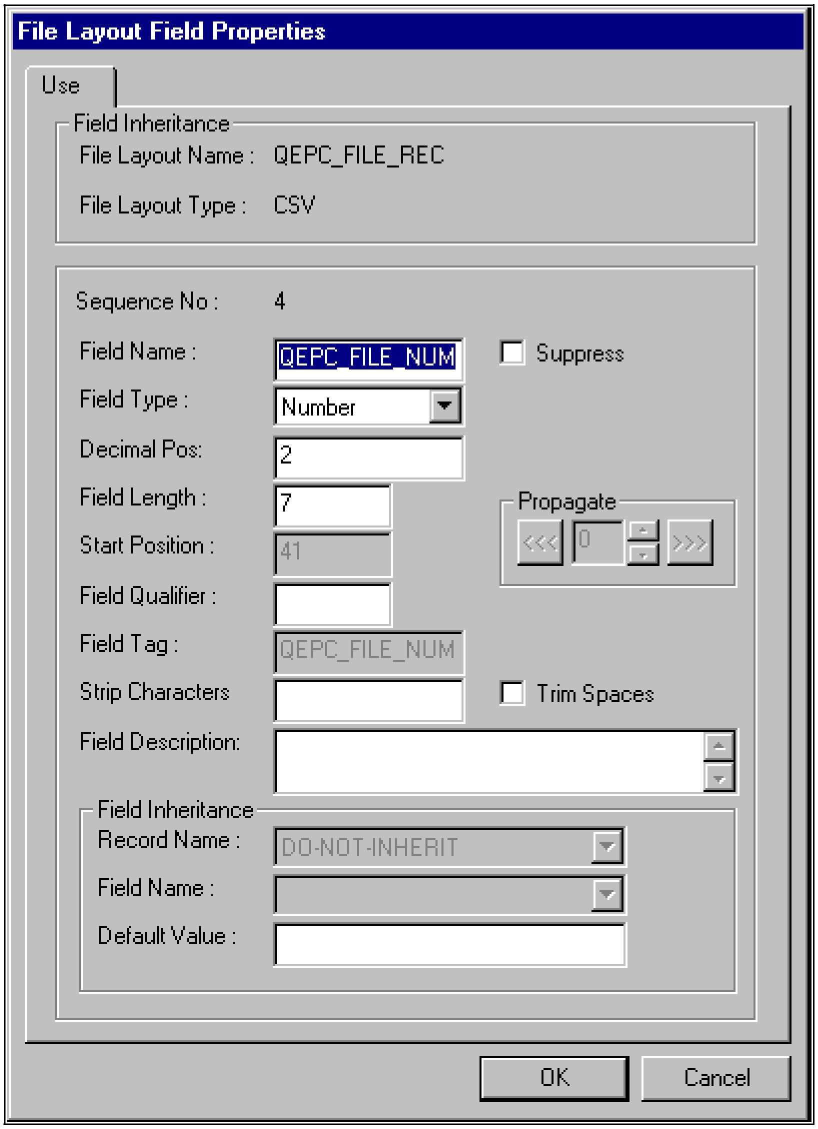
The properties for the QEPC_FILE_REC File Layout are as follows:



File Layout Definition Properties

Note that the Definition Qualifier is double-quotes ("), while the separator is a comma.

Remember, the last field didn't have a qualifier. To account for that, the field properties for this field must be edited, and a blank must be put in the Field Qualifier property.



File layout Field Properties

See *Enterprise PeopleTools 8.51 PeopleBook: Application Engine*, "Understanding Application Engine."

Multiple File Layouts

In the previous examples, the input file contained rowsets based on a single File Layout definition. However, PeopleTools provides the functionality to process input files containing rowsets that require several *different* File Layouts.

Note. You can use only fixed format files to implement multiple file layouts.

See the SetFileId method for details about handling multiple file layouts.

In this section, we discuss how to:

- Read multiple file layouts.
- Write multiple file layouts.

See Chapter 19, "File Class," SetFileId, page 996.

Reading Multiple File Layouts

If your input file contains data based on more than one File Layout, it must contain an indicator, called a FileId that specifies:

- When a different File Layout definition should be used.
- Which File Layout definition should be used.

The FileId must be specified on a separate line and must precede *every* rowset that requires a layout different from the previous rowset. It isn't considered part of the rowset.

In the following example, the file contains two FileId lines; they use a file record ID that distinguishes them from the rowset data—in this case, "999".

```
999 PRODUCT  /* The following rowset uses the PRODUCT layout */
001  /* Level 0 record data */
101  /* level 1 record data */
201  /* Level 2 record data */
201  /* Level 2 record data */
999 ORDER  /* The following two rowsets use the ORDER layout */
001  /* Level 0 record data */
111  /* Level 1 record data */
111  /* Level 1 record data */
001  /* Level 0 record data */
111  /* Level 1 record data */
111  /* Level 1 record data */
```

The FileId can contain any information you want that indicates which file layout to use; the "PRODUCT" and "ORDER" fields shown are just examples.

To read this file, you should do the following in your program:

1. Use the SetFileId method to specify the file record ID value.
2. Use the ReadRowset method to read the data.
3. Check if the rowset is NULL.

   NULL indicates you've reached either the end of the file or a new rowset.
4. Use the IsNewFileId property to check if the next line is a FileId file record (line).

- If IsNewFileld is False, you've reached the end of the file.
- If IsNewFileld is True, use the CurrentRecord property to determine which File Layout to use next.

The following example reads rowsets from a file. When it finds a new rowset (indicated by &IsNewFileld returning True) the value of &CurrentRecord is passed to a function that reads and evaluates the line, then returns the name of the new file layout. (The code for the function FindFileId is included at the start of the example.)


```
Local File &MYFILE;
Local Rowset &rsFile;
Local Record &rSomeRec1, &rSomeRec2;
Local SQL &SQL1;

Function FindFileID(&CurrentRecord As string) Returns string ;
   Evaluate RTrim(Substring(&CurrentRecord, 5, 50))
   When "SOME_REC1"
      &FILELAYOUT = "SOME_REC1";
   When "SOME_REC2"
      &FILELAYOUT = "SOME_REC2";
   End-Evaluate;
   Return &FILELAYOUT;
End-Function;

&rSomeRec1 = CreateRecord(Record.SOME_REC1);
&rSomeRec2 = CreateRecord(Record.SOME_REC2);
&SQL1 = CreateSQL("%Insert(:1)");
&MYFILE = GetFile("c:\temp\MULTI_FILE.out", "R", %FilePath_Absolute);

rem Set temporary first file layout;
&MYFILE.SetFileLayout(FileLayout.SOME_REC1);
&MYFILE.SetFileId("999", 1);

rem Read rowset to find actual first file ID;
&rsFile = &MYFILE.ReadRowset();
&CurrentRecord = &MYFILE.CurrentRecord;
&IsNewFileID = &MYFILE.IsNewFileId;
If &MYFILE.IsNewFileId Then
   &FILELAYOUT = FindFileID(&CurrentRecord);
   &MYFILE.SetFileLayout(@("FileLayout." | &FILELAYOUT));
   &MYFILE.SetFileId("999", 1);
End-If;

rem Read first 'real' rowset;
&rsFile = &MYFILE.ReadRowset();
&CurrentRecord = &MYFILE.CurrentRecord;
&IsNewFileID = &MYFILE.IsNewFileId;
While &rsFile <> Null Or
      &IsNewFileID
   If &MYFILE.IsNewFileId Then
      &FILELAYOUT = FindFileID(&CurrentRecord);
      &MYFILE.SetFileLayout(@("FileLayout." | &FILELAYOUT));
      &MYFILE.SetFileId("999", 1);
      If &IsNewFileID Then
         &rsFile = &MYFILE.ReadRowset();
         &CurrentRecord = &MYFILE.CurrentRecord;
         &IsNewFileID = &MYFILE.IsNewFileId;
      End-If;
   End-If;

   Evaluate &FILELAYOUT

   When "SOME_REC1"
      &rsFile(1).SOME_REC1.CopyFieldsTo(&rSomeRec1);
      &rSomeRec1.ExecuteEdits(%Edit_Required);
      If Not &rSomeRec1.IsEditError Then
         &SQL1.Execute(&rSomeRec1);
      End-If;
      Break;
   When "SOME_REC2"
      &rsFile(1).SOME_REC2.CopyFieldsTo(&rSomeRec2);
      &rSomeRec2.ExecuteEdits(%Edit_Required);
      If Not &rSomeRec2.IsEditError Then
```



```
            &SQL1.Execute(&rSomeRec2);
         End-If;
      End-Evaluate;

      &rsFile = &MYFILE.ReadRowset();
      &CurrentRecord = &MYFILE.CurrentRecord;
      &IsNewFileID = &MYFILE.IsNewFileId;
   End-While;

   &MYFILE.Close();
```

See Chapter 19, "File Class," SetFileId, page 996; Chapter 19, "File Class," ReadRowset, page 994 and Chapter 19, "File Class," IsNewFileId, page 1011.

Writing Multiple File Layouts

If you're writing files that contain data based on more than one File Layout definition, consider the following points:

- If the file is going to a third-party vendor, you should work with the third-party to determine what their requirements are for specifying the different data formats.
- If the file is going to be used by another PeopleSoft system, you must add the FileId between each rowset that requires a different layout. FileId file records are *not* part of any rowset. They should be designed so they won't be mistaken for part of a rowset. You can create and write them to the file in many ways. The following are suggestions:
  - Build each line as a string, using any of the built-in string manipulation functions, then write them to the file using the File class WriteLine or WriteString methods.
  - Design a file layout consisting of a single file record definition for the FileId file records, then build the records using Record Class methods and functions, and write them to the file using the WriteRecord method.

The following code example writes each record from the level one scroll on a page to the file using a different File Layout. Between each WriteRowset the File ID file record is written to the file, describing the new File Layout being used.


```
Local File &MYFILE;
Local Rowset &FILEROWSET;
Local Record &REC1, &REC2;
Local SQL &SQL;

&MYFILE = GetFile("c:\temp\Records.txt", "W", %FilePath_Absolute);
If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FileLayout.TREE_LEVEL) Then
      &REC1 = CreateRecord(Record.PSTREELEVEL);
      &FILEROWSET = &MYFILE.CreateRowset();
      &SQL = CreateSQL("%Selectall(:1)", &REC1);
      /* write first File ID to file */
      &MYFILE.WriteLine("999 FILE LAYOUT 1");
      While &SQL.Fetch(&REC1)
         &REC1.CopyFieldsTo(&FILEROWSET.GetRow(1).PSTREELEVEL);
         &MYFILE.WriteRowset(&FILEROWSET, True);
      End-While;
   Else
      /* file layout not found, do error processing */
   End-If;

   If &MYFILE.SetFileLayout(FileLayout.TREE_USERLEVEL) Then
      &REC2 = CreateRecord(Record.TREE_LEVEL_TBL);
      &FILEROWSET = &MYFILE.CreateRowset();
      &SQL = CreateSQL("%Selectall(:1)", &REC2);
      /* write second File ID to file */
      &MYFILE.WriteLine("999 FILE LAYOUT 2");
      While &SQL.Fetch(&REC2)
         &REC2.CopyFieldsTo(&FILEROWSET.GetRow(1).TREE_LEVEL_TBL);
         &MYFILE.WriteRowset(&FILEROWSET);
      End-While;
   Else
      /* file layout not found, do error processing */
   End-If;
Else
   /* file not opened, do error processing */
End-If;
&MYFILE.Close();
```

See Chapter 19, "File Class," WriteLine, page 1002; Chapter 19, "File Class," WriteString, page 1008 and Chapter 19, "File Class," WriteRecord, page 1004.

See Chapter 34, "Record Class," page 2077.

Chapter 20

Grid Classes

This chapter provides an overview of Grid class and the GridColumn subobject and discusses the following topics:

- Shortcut considerations
- The Grid class in PeopleCode
- Grid or Grid Column declaration
- Scope of a Grid or Grid Column class
- Grid Class built-in function
- Grid Class reference
- Grid Column class

Note. Use the grid classes to control the display of a grid control. If you want to manipulate an analytic grid, used with PeopleSoft Analytic Calculation Engine data, you must use the AnalyticGrid classes.

See Also

Chapter 4, "Analytic Grid Classes," page 159

Understanding Grid and GridColumn Classes

A grid looks and behaves like a spreadsheet embedded in a page: it has column headings, row headings, cells, and horizontal and vertical scroll bars. It can be used instead of a single-level scroll. It is analogous to a scroll region on a page. Each row in a grid corresponds to a set of controls in a scroll occurrence. Each cell in a grid corresponds to a field on a page.

The grid class enables you to instantiate only GridColumn objects, which in turn enables you to change grid column attributes without having to write PeopleCode that loops through every row of the grid.

Note. PeopleSoft builds a page grid one row at a time. Because the grid class applies to a complete grid, you can't attach PeopleCode that uses the grid class to events that occur before the grid is built; the earliest event you can use is the page Activate Event.

See Also

Chapter 20, "Grid Classes," GridColumn Class, page 1045

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Using Scroll Areas, Scroll Bars, and Grids," Using Grids

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Activate Event

Shortcut Considerations

An expression of the form

`&MYGRID.`*`columnname.property`*

is converted to an object expression by using**`GetColumn`***`(columnname)`*. The following examples are equivalent.

Using *columnname:*

```
&MYGRIDCOLUMN = &MYGRID.CHECKLIST_ITEMCODE;
```

Using the GetColumn method:

```
&MYGRIDCOLUMN = &MYGRID.GetColumn("CHECKLIST_ITEMCODE");
```

The following examples are equivalent.

Using *columnname*:

```
&MYGRID.CHECKLIST_ITEMCODE.Visible = False;
```

Using the GetColumn method:

```
&MYGRID.GetColumn("CHECKLIST_ITEMCODE").Visible = False;
```

The Grid Class in PeopleCode

The PeopleCode grid object is a reference to a page runtime object for the grid. These particular page runtime objects aren't present until the Component is started.

Note. PeopleSoft builds a page grid one row at a time. Because the Grid class applies to a complete grid, you can't attach PeopleCode that uses the Grid class to events that occur before the grid is built; the earliest event you can use is the page Activate Event.

If you're using the grid within a secondary page, the runtime object for the grid isn't created until the secondary page is run. The grid object can't be obtained until then, which means that the earliest PeopleCode event you can use to activate a grid that's on a secondary page is the Activate event for the secondary page.

The attributes you set for displaying a page grid remain in effect only while the page is active. When you switch between pages in a component, you have to reapply those changes *every time* the page is displayed.

In addition, the Activate event associated with a page fires every time the page is displayed. Any PeopleCode associated with that Activate event runs, which may undo the changes you made when the page was last active. For example, if you hide a grid column in the Activate event, then display it as part of a user action, when the user tabs to another page in the component, then tabs back, the Activate event runs again, hiding the grid column again.

If a user at runtime hides a column of a grid, tabs to another page in the component, then tabs back to the first page, the page is refreshed and the grid column is displayed again.

When you place a grid on a page, the grid is automatically named the same as the name of the primary record of the scroll for the grid. This is the name you use with the GetGrid function. You can change this name on the Record tab of the Grid properties.

Note. There is no visible property for a grid, just a grid column. However, you can still hide an entire grid. Remember, many of the Rowset methods and properties work on a grid. If you want to hide an entire grid, get the rowset for that grid by using the HideAllRows Rowset class method.

Use the grid classes to access an ordinary grid. Use the analytic grid classes to access an analytic grid.

See Also

Chapter 4, "Analytic Grid Classes," page 159

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Activate Event

Chapter 36, "Rowset Class," HideAllRows, page 2147

Data Type for a Grid or Grid Column Object

Grids are declared using the Grid data type. For example,

```
Local Grid &MYGRID;
```

Grid Columns are declared using the GridColumn data type. For example:

```
Local GridColumn &MYGRIDCOL;
```

Scope of a Grid or Grid Column Object

Both the grid and grid column objects can be instantiated from PeopleCode only.

A grid is a control on a page. You generally use these objects only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message subscription, a Component Interface, and so on.

In addition, PeopleSoft builds a page grid one row at a time. Because the Grid class applies to a complete grid, you can't attach PeopleCode that uses the Grid class to events that occur before the grid is built; the earliest event you can use is the page Activate Event.

Grid Class Built-in Function

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetGrid

Grid Class Methods

In this section, we discuss each Grid class method.

EnableColumns

Syntax

`EnableColumns`*`(&Array)`*

Description

Use this method to enable or disable one or more columns in a grid. When a column is enabled, it is editable; when it's disabled, it is un-editable.

The EnableColumns method of the Grid class can provide a noticeable performance improvement over multiple calls to set the Enabled property of the GridColumn class. Each call manipulating a grid (either using a Grid class method or setting a GridColumn property) has a significant, and similar performance overhead. Therefore, one key to increasing the performance of PeopleCode programs manipulating grids is to reduce the number of these calls. Performance testing data suggests that if your program is changing three or more columns, use one of the Grid class methods, such as the EnableColumns method, instead of setting the column properties directly. The single call to the Grid class method offsets the small overhead of creating and populating the required array.

Parameters

Parameter	***Description***
&Array	Specify a two-dimensional array containing the column name and a value (Y/N) indicating whether the column is enabled.

Returns

None.

Example

```
&AREnable = CreateArrayRept(CreateArrayRept("", 2), 0);
&AREnable.Push(CreateArray("JOB_DETAIL", "Y"));
&AREnable.Push(CreateArray("JOB_TIME", "Y"));
&myGrid.EnableColumns(&AREnable);
```

See Also

Chapter 20, "Grid Classes," Enabled, page 1046

GetColumn

Syntax

GetColumn*(columnname)*

Description

The GetColumn method instantiates a grid column object from the Grid class, and populates it with a grid column from the grid object executing this method. Specify the grid column name in the page field properties for that field, consisting of any combination of uppercase letters, digits and "#", "$", "@", and "_".

To specify a grid column name:

1. Open the page in Application Designer, select the grid and access the page field properties.
2. Select the Columns tab on the grid properties.
3. Either double-click on the grid column you want to name, or select the column and click the Properties button.
4. On the General tab, type the grid column name in the Page Field Name field.

Note. Although it's possible to base multiple grid columns on the same record field, the Page Field Name you enter for each grid column must be unique, not just for that grid, but for that page. If you have two grids on a page, every page field name must be unique to the page.

Parameters

Parameter	***Description***
columnname	Specify a string containing the value of the Page Field Name on the General tab of the grid column's properties. You must have previously entered a value for the Page Field Name for the grid column you want to work with.

Returns

A GridColumn object populated with the grid column specified as the parameter to this method.

Example

```
local Grid &MYGRID;
local GridColumn &MYGRIDCOLUMN;

&MYGRID = GetGrid(PAGE.EMPLOYEE_CHECKLIST, "EMPL_GRID");
&MYGRIDCOLUMN = &MYGRID.GetColumn("CHECKLIST_ITEMCODE");
```

The following function loops through rows on a grid. The function finds each row that is selected. It does this through the Selected property of the Row class of PeopleCode. Data is then moved from the selected row to a new row, on a different grid, in the same page. The way in which this function is written, data is moved from &SCROLL_SHELF to &SCROLL_CART. These are two different rowset objects, of two different grids, on the same page. Note that the two grids in this example are on the same occurs level.


```
/* Moving data between grids on the same occurs level */
/* of the same page */

Local Rowset &SCROLL_CART, &SCROLL_SHELF;

Function move_rows(&SCROLL_CART As Rowset, &SCROLL_SHELF As Rowset);

   &I = 1;
/* loop to find whether row is selected */
   Repeat
   If &SCROLL_SHELF.GetRow(&I).Selected = True Then

      If All(&SCROLL_CART(1).GetRecord(1).QEPC_ITEM.Value) Then

         &SCROLL_CART.InsertRow(&SCROLL_CART.ActiveRowCount);
      End-If;

   /* if it is selected move data to other grid   */

      &SCROLL_SHELF.GetRow(&I).GetRecord(1).CopyFieldsTo(&SCROLL_CART.GetRow⇒
(&SCROLL_CART.ActiveRowCount).GetRecord(1));

/* delete row from current grid so data disappears  */

      &SCROLL_SHELF.DeleteRow(&I);
      &I = &I - 1;
   End-If;

   &I = &I + 1;
   Until &I = &SCROLL_SHELF.ActiveRowCount + 1;

/* end of loop ****************************************/

End-Function;

/********* end of function *****************************/

/* Creating the rowset object */

&SCROLL_CART = GetLevel0()(1).GetRowset(SCROLL.QEPC_CART);
&SCROLL_SHELF = GetLevel0()(1).GetRowset(SCROLL.QEPC_SHELF);

/* calling the function   */

move_rows(&SCROLL_CART, &SCROLL_SHELF);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetGrid

LabelColumns

Syntax

LabelColumns*(&Array)*

Description

Use this method to set the display label for one or more columns in a grid.

The LabelColumns method of the Grid class can provide a noticeable performance improvement over multiple calls to set the Label property of the GridColumn class. Each call manipulating a grid (either using a Grid class method or setting a GridColumn property) has a significant, and similar performance overhead. Therefore, one key to increasing the performance of PeopleCode programs manipulating grids is to reduce the number of these calls. Performance testing data suggests that if your program is changing three or more columns, use one of the Grid class methods, such as the LabelColumns method, instead of setting the column properties directly. The single call to the Grid class method offsets the small overhead of creating and populating the required array.

Parameters

Parameter	***Description***
&Array	Specify a two-dimensional array containing the column name and a value for the column label.

Returns

None.

Example

```
&ARLabel= CreateArrayRept(CreateArrayRept("", 2), 0);
&ARLabel.Push(CreateArray("JOB_DETAIL", "Job Detail"));
&ARLabel.Push(CreateArray("JOB_TIME", "Job Time"));
&myGrid.LabelColumns(&ARLabel);
```

See Also

Chapter 20, "Grid Classes," Label, page 1047

SetProperties

Syntax

SetProperties*(&Array)*

Description

Use this method to set multiple properties (column enabled, column visibility, and column label) for one or more columns in a grid.

The SetProperties method of the Grid class can provide a noticeable performance improvement over multiple calls to set individual properties of the GridColumn class. Each call manipulating a grid (either using a Grid class method or setting a GridColumn property) has a significant, and similar performance overhead. Therefore, one key to increasing the performance of PeopleCode programs manipulating grids is to reduce the number of these calls. Performance testing data suggests that if your program is changing three or more columns, use one of the Grid class methods, such as the SetProperties method, instead of setting the column properties directly. The single call to the Grid class method offsets the small overhead of creating and populating the required array.

Parameters

Parameter	***Description***
&Array	Specify a four-dimensional array containing the column name, a value (Y/N) indicating whether the column is enabled, a value (Y/N) indicating whether the column is visible, and a value for the column label.

Returns

None.

Example

```
&ARProp= CreateArrayRept(CreateArrayRept("", 4), 0);
&ARProp.Push(CreateArray("JOB_DETAIL", "Y", "Y", "Job Detail"));
&ARProp.Push(CreateArray("JOB_TIME", "Y", "Y", "Job Time"));
&mygrid.SetProperties(&ARProp);
```

See Also

Chapter 20, "Grid Classes," Enabled, page 1046; Chapter 20, "Grid Classes," Label, page 1047 and Chapter 20, "Grid Classes," Visible, page 1048

ShowColumns

Syntax

`ShowColumns`*`(&Array)`*

Description

Use this method to set the visibility for one or more columns in a grid.

The ShowColumns method of the Grid class can provide a noticeable performance improvement over multiple calls to set the Visible property of the GridColumn class. Each call manipulating a grid (either using a Grid class method or setting a GridColumn property) has a significant, and similar performance overhead. Therefore, one key to increasing the performance of PeopleCode programs manipulating grids is to reduce the number of these calls. Performance testing data suggests that if your program is changing three or more columns, use one of the Grid class methods, such as the ShowColumns method, instead of setting the column properties directly. The single call to the Grid class method offsets the small overhead of creating and populating the required array.

Parameters

Parameter	***Description***
&Array	Specify a two-dimensional array containing the column name and a value (Y/N) indicating whether the column is visible.

Returns

None.

Example

```
&ARShow= CreateArrayRept(CreateArrayRept("", 2), 0);
&ARShow.Push(CreateArray("JOB_DETAIL", "Y"));
&ARShow.Push(CreateArray("JOB_TIME", "Y"));
&myGrid.ShowColumns(&ARShow);
```

See Also

Chapter 20, "Grid Classes," Visible, page 1048

Grid Class Properties

In this section, the Grid class properties are described in alphabetical order.

gridcolumn

Description

If a grid column name is used as a property, it accesses the grid column with that name. This means the following code:

```
&Mycolumn = &MyGrid.gridcolumnname;
```

acts the same as


```
&Mycolumn = &MyGrid.GetColumn(columnname);
```

This property is read-only.

Label

Description

This property returns a string specifying the label that appears as the title of the grid.

Note. You can't use this property to set labels longer than 100 characters. If you try to set a label of more than 100 characters, the label is truncated to 100 characters. Always put any changes to labels in the Activate event. This way the label is set every time the page is accessed.

This property is read-write.

SummaryText

Description

Use this property to set or return a string representing the summary text for the grid.

Summary text enables you to provide a brief description of the functionality and content of the grid area. This property is pertinent for users who access the application in accessibility mode using screen readers.

This property is read-write.

Example

```
&MyGrid = GetGrid(Page.PSXLATMAINT, "PSXITMMNT_VW");
&MyGrid.SummaryText = "This is the new summary text through PeopleCode";
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Using Scroll Areas, Scroll Bars, and Grids," Setting Grid Label Properties

GridColumn Class

Grid columns are page fields that comprise a grid, which is itself a page field. Similarly, a group of GridColumn objects comprises a Grid object. The GridColumn class enables you to change grid column attributes without having to write PeopleCode that loops through every row of the grid.

This class has no methods or built-in functions, only properties.

Note. To use a GridColumn object, you must instantiate it from a grid object. This requires you to first instantiate a Grid object using the GetGrid built-in function.

You create an analytic grid column object from the analytic grid class. However, the analytic grid column class has the same properties as the grid column class. Any differences are noted in the documentation.

See *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetGrid.

GridColumn Class Properties

In this section, we discuss each GridColumn class property.

Enabled

Description

This property specifies whether the fields in the column are enabled (that is, can be edited) or if they are "disabled", that is, un-editable. All columns are enabled by default.

Note. For an analytic grid column, the Enabled property only works with cubes, not dimensions.

This property is read-write.

Example

```
If Not &ValidID Then
   &MYGRID = GetGrid(PAGE.EMPLOYEE_CHECKLIST, "EMPL_GRID");
   &MYGRIDCOLUMN = &MYGRID.GetColumn("CHECKLIST_ITEMCODE");
   &MYGRIDCOLUMN.Enabled = False;
End-If;
```

See Also

The EnableColumns and SetProperties methods of the Grid class can provide a noticeable performance improvement over multiple calls to set the Enabled property of the GridColumn class.

Chapter 20, "Grid Classes," EnableColumns, page 1038 and Chapter 20, "Grid Classes," SetProperties, page 1042

Label

Description

This property specifies the display label for the GridColumn object , as distinct from the grid column's Record Field Name or Page Field Name. This property overrides the equivalent settings on the Label tab of the grid column's Page Field Properties.

Note. You can't use this property to set labels longer than 100 characters. If you try to set a label of more than 100 characters, the label is truncated to 100 characters. If you change the column label, then change the field label, you may overwrite the column label with the field label. The grid object is part of a page, *not* the data buffers, while the field is part of the data buffers. To avoid this problem, always put any changes to column labels in the Activate event. This way the label is set every time the page is accessed.

This property is read-write.

Example

```
&MYGRIDCOLUMN.Label = "Checklist Item";
```

See Also

The LabelColumns and SetProperties methods of the Grid class can provide a noticeable performance improvement over multiple calls to set the Label property of the GridColumn class.

Chapter 20, "Grid Classes," LabelColumns, page 1041 and Chapter 20, "Grid Classes," SetProperties, page 1042

Name

Description

This property returns the name of the grid column, as a string. This value comes from the Page Field Name on the General tab in the Page Field Properties of the GridColumn object executing the property. This property was the value used to instantiate the GridColumn object.

This property is read-only.

Example

```
&PF_NAME = &MYGRIDCOLUMN.Name;
```


Visible

Description

This property specifies whether a grid column is visible or hidden. Set this property to False to hide the grid column, and to True to unhide the grid column. This property defaults to True.

The Visible property also hides grid columns that are displayed as tabs in the PeopleSoft Pure Internet Architecture.

If you specify "Show Column on Hide Rows" in Application Designer, the column headers and labels of a grid display at runtime, even when the rest of the column is hidden. You can't override this value using PeopleCode.

Note. For an analytic grid column, this property is only valid if the column is bound to a dimension on a slicer, and bound to a cube that is on the column axis.

This property is read-write.

Note. In previous releases of PeopleTools, the methodology for hiding a grid column was to use the Hide function in a loop to hide each cell in the column, one row at a time. This method of hiding grid columns still works. However, your application will experience deteriorated performance if you continue to use this method.

Example

```
&MYGRIDCOLUMN.Visible = False;
```

The following example checks for the value of a field in every row of the grid. If that value is "N" for every row, the column is hidden.


```
&RS = GetRowset(Scroll.EX_SHEET_LINE);
&HIDE = True;
While (&HIDE)

   For &I = 1 To &RS.ActiveRowCount;
      &OUT_OF_POLICY = &RS(&I).EX_SHEET_LINE.OUT_OF_POLICY.Value;
      &NO_RECEIPT_FLG = &RS(&I).EX_SHEET_LINE.NO_RECEIPT_FLG.Value;
      If &OUT_OF_POLICY = "Y" Then
         &OUT_OF_POLICY_HIDE = False;
         &HIDE = False;
      End-If;

      If &NO_RECEIPT_FLG = "Y" Then
         &NO_RECEIPT_HIDE = False;
         &HIDE = False;
      End-If;
   End-For;

   If &HIDE = True Then
      &HIDE = False;
   End-If;

End-While;

If Not (&OUT_OF_POLICY_HIDE) Then
   GetGrid(Page.EX_SHEET_LINE_APV1, "EX_SHEET_LINE").OUT_OF_POLICY.Visible = True;
Else
   GetGrid(Page.EX_SHEET_LINE_APV1, "EX_SHEET_LINE").OUT_OF_POLICY.Visible = False;
End-If;

If Not (&NO_RECEIPT_HIDE) Then
   GetGrid(Page.EX_SHEET_LINE_APV1, "EX_SHEET_LINE").NO_RECEIPT_FLG.Visible = True;
Else
   GetGrid(Page.EX_SHEET_LINE_APV1, "EX_SHEET_LINE").NO_RECEIPT_FLG.Visible =⇒
 False;
End-If;
```

See Also

The ShowColumns and SetProperties methods of the Grid class can provide a noticeable performance improvement over multiple calls to set the Visible property of the GridColumn class.

Chapter 20, "Grid Classes," ShowColumns, page 1043 and Chapter 20, "Grid Classes," SetProperties, page 1042

Chapter 21

Internet Script Classes (iScript)

This chapter describes how an Internet Script (iScript) works in your application, how to create an iScript, and the Internet Script classes that you use in your PeopleCode. It also discusses the following topics:

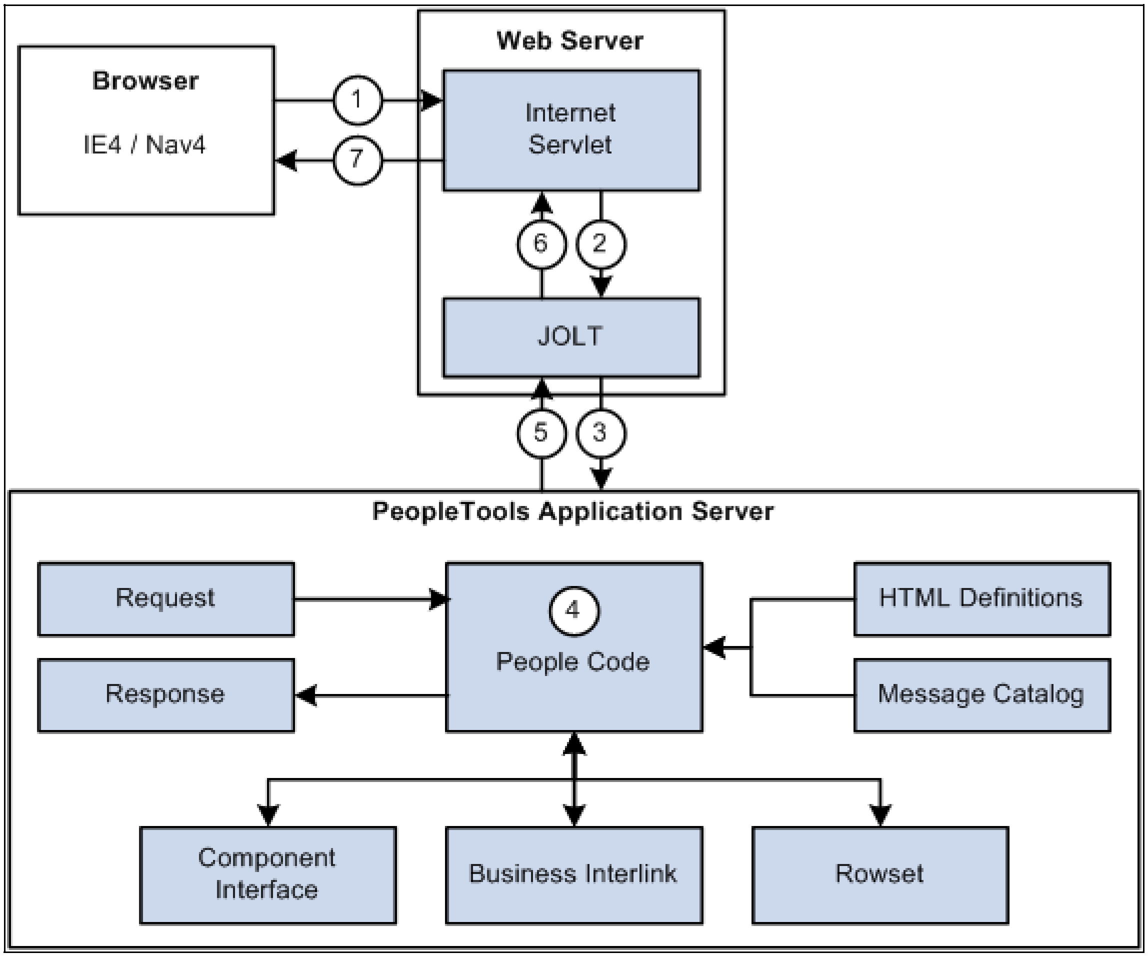
- URL vs. URI.
- Web libraries.
- iScript security.
- When to use an iScript.
- Details of an iScript URL.
- The Generate functions.
- Error handling.
- Scope of the Internet Script classes.
- Data Types of the Internet Script classes.
- Internet Script classes built-in functions.
- Internet Script reference.

Understanding Internet Script Classes

An Internet Script, also called an iScript, is a specialized PeopleCode function that generates dynamic web content. iScripts interact with web clients (browsers) using a request-response paradigm based on the behavior of the Hypertext Transfer Protocol.

iScripts work with PeopleSoft Pure Internet Architecture. You must have PeopleSoft Pure Internet Architecture set up correctly before you can run an iScript.

The following flow chart shows the interaction of an iScript (PeopleCode) in a PeopleSoft Pure Internet Architecture application.



Communication flow chart for Internet Scripts

1. An HTTP request arrives at the web server specifying in its URL the Presentation Relay Servlet (psc). The path information included in the URL specifies which PeopleCode program to run. No component is specified because none is needed. The PeopleCode program (iScript) runs directly. The following is an example of the URL:

   ```
   http://serverx/Servlets/psp/ps84/e_procurement/fdm/s/WEBLIB_BEN_401k.PAGES.Field⇒
   Formula.IScript_Home401k
   ```

2. The psc servlet is invoked by the web server. It serializes all necessary information from the web server request and response objects, then makes a call through JOLT to the PeopleSoft application server service passing the serialized object information.

3. The application server unpacks all of the object information and creates PeopleSoft versions of the Request and Response objects. It then calls the function in the PeopleCode program specified in the URL. The PeopleCode Request and Response objects are available to the PeopleCode program through the system variables %Request and %Response.

4. The PeopleCode program is then responsible for generating every aspect of the HTML page that is to be returned to the browser. It has access to the Request object to view items such as query string parameters, cookies, and headers. It also has access to component interfaces, business interlinks, and Rowset objects to interact with applications. HTML definitions are available to include language sensitive blocks of HTML. They can also contain JavaScript. The message catalog is also available for language sensitive messages. The Response object is used to "write" the generated HTML back to the browser.

5. When the PeopleCode program is finished running, the headers, cookies, and HTML in the PeopleSoft Response object are serialized and returned through JOLT to the Presentation Relay Servlet.

6. The Presentation Relay Servlet then unpacks the Response information generated by the PeopleCode.

7. The servlet plays the information back to the web server response object, which in turn sends the response to the user's browser.

URL vs. URI

In this document, the term URL refers to the entire URL, that points to content. The following is an example of a URL:

```
http://someserver/servlets/psc/ps84/e_procurement/fdm/s/WEBLIB_Portal.Field⇒
Change.IScript_DoSomething?page=view&key1=value1&key2=value2
```

A URI does not include the content information. Think of it as a subset of the URL that points to the location of the resource, but does not include any parameters passed to that resource. From the previous example, the URI portion of the URL is:

```
http://someserver/servlets/psc/ps84
```

Web Libraries

iScripts use the existing PeopleCode Function Library infrastructure. However, instead of naming your record FUNCLIB_xxx, all iScripts *must* be contained in records named WEBLIB_xxx. All of the existing tools and techniques for working with function libraries (such as Upgrade, Find In, Rename, and so on) also apply to Web Libraries.

See *PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide*, "Using the PeopleCode Editor," Accessing PeopleCode External Functions.

To create an iScript:

1. Create or extend a domain specific WEBLIB (funclib).

   WEBLIBs are derived/work record definitions that have their name prefixed with WEBLIB_.

2. Add a function to a WEBLIB.

   The name of your function *must* be prefaced with IScript_. For example:

   ```
   IScript_HelloWorld
   ```

   ```
   IScript_FuncHRPage
   ```

Note. iScript functions take no arguments, and do not return a value.

The following iScript writes data.


```
Function IScript_HPDefaultCategories()

   &ClearDotImage = %Response.GetImageURL(Image.PT_PORTAL_CLEAR_DOT);
   &CatHTML = GetHTMLText(HTML.PORTAL_HP_CATEGORY, &ClearDotImage, GetCategories⇒
());
   %Response.Write(&CatHTML);

End-Function;
```

The following iScript uses both the request and response objects to echo what the user types into an edit control.

Example iScript

```
Function IScript_HelloWorld()

%Response.WriteLine("<html><head><title>Hello World</title></head><body>");
   %Response.WriteLine("<h1>Hello World</h1>");
   %Response.WriteLine("<form method=POST action = " | %Request.FullURI | "?" |⇒
 %Request.QueryString | "><input type=submit value='Say this!'> <input type=⇒
edit name=WhatYouSaid value=" | %Request.GetParameter("WhatYouSaid") | "><⇒
/form>");

   rem echo back what the user typed into the edit box...;
   %Response.WriteLine("<p><font face='Arial, Helvetica' size='5' color='blue'⇒
 >You said: ");
   %Response.WriteLine(%Request.GetParameter("WhatYouSaid"));

   %Response.WriteLine("</body></html>");
   Return;
End-Function;
```

iScript Security

iScripts are secured on your system similar to Component Interfaces. After you create a WEBLIB record, and your functions, you must add them just as you would add a page to an application.

To add security to an iScript :

1. Navigate to the Maintain Security page and open the Permission List to which you want to give access.
2. Select Web Libraries.

   You may need to scroll through the tabs at the top of the page to access the Web Libraries tab.
3. (Optional) Add new WebLibrary for Permission List.

   If this is a new Web Library, you must add it to the permission list. Click the plus button to add a new Web Library, then select the library you want to add from the drop-down list box.
4. Select Edit for the Web Library you want to grant access for.
5. Select the access you want to give each function in the WEBLIB record.

   From the page that displays, you can choose to allow or disallow access on a per function basis. The buttons on the side of the page either grant or disallow access for all functions.

See *PeopleTools 8.51 PeopleBook: PeopleSoft Security Administration*

When to Use an iScript

iScripts give you complete control over the HTML sent to the browser. This enables flexibility in creating a user interface. However, there are some responsibilities that you inherit when choosing to use iScripts.

- You are responsible for creating HTML and JavaScript that is cross-browser compatible.
- You are responsible for ensuring that all the text sent to the browser is stored in either the message catalog or in HTML definitions to enable translation of the text.
- iScripts aren't part of the regular Component Processor flow, so you are responsible for accessing the database in a multi-language, multi-market, and multi-currency sensitive way. (Do this by using a Component Interface to access the database.)

For these reasons, PeopleSoft recommends that you first try to build a page in Application Designer. This approach enables PeopleTools to generate all of the HTML in a cross-browser, multi-language, multi-market, and multi-currency sensitive way.

You don't have to use an iScript to generate an entire page. You can use the Request and Response objects with an HTML area, so you can develop a portion of your HTML using the iScript objects, while allowing the majority of it to be generated by the Component Processor. If you use this method, instead of using the Response object's Write and WriteLine methods to output your HTML, you set the value of the HTML area to the HTML that you want to display. Remember, that just as with an iScript, you must ensure that the HTML you generate is cross-browser compatible and multi-market sensitive.

So when *should* you use iScripts? Here are two scenarios where iScripts would be an appropriate choice:

- The page being developed *cannot* be built using Application Designer. An example of this is a page that requires more than one HTML form. PeopleSoft Pure Internet Architecture places the entire page inside of a single form tag, so no other HTML form tags can be added. In this case the requirements of the page can't be met by pages created in Application Designer, so use iScripts instead.

  Note. You should use Component Interfaces for all database access so you have multi-language and multi-currency sensitivity.

- The page being developed never accesses the database. Using a page and Component Processor for this type of page incurs unnecessary processing overhead. An example of this is a page that talks to another website and redisplays HTML from the remote site.

Style Sheets and Styles

PeopleSoft recommends that developers using the iScripts always use styles (also known as classes) defined in the style sheets to specify the attributes (that is, background color, font, size, alignment, borders, and so on) of objects referenced in the iScripts. The Response object provides access to Style Sheets stored in the PeopleSoft database.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Style Sheet Definitions"

Other Considerations

PeopleSoft does not recommend using the following technologies in iScripts:

- Java applets
- Browser plug-ins

Details of an iScript URL

Viewing an iScript requires the assembly of a URL with the following pieces:

Section	***Description***
http://Server/	The scheme (http / https) and web server name.
servlet_name/	The name of the physical servlet that the web server invokes to handle the request. This is either psp or psc.
SiteName/	The user-defined site name. This is defined during the installation of PIA. This enables you to set up multiple sites on one physical web server.
PortalName/	The name of the portal to use for this request. The portal object contains metadata that describes how to present the content (that is, template, pagelets and so on.)

Section	***Description***
NodeName/	The name of the node that contains the content for this request.
content_type/	The type of the content for this request. For iScripts, this is "s".
content_id	The identification of the content. This, and the type is the unique key to the content being retrieved.
?content_parm	The query string parameters (name value pairs) for the content.

For an iScript, the *content_id* has the following form:

```
Record.Field.Event.Function
```

The following is an example of the call to an iScript:

```
http://mlee2038//psp/PS84/e_procurement/fdm/s/WEBLIB_Portal.PORTAL_HEADER.Field⇒
Formula.IScript_UniHeader_PIA
```

The Generate Functions

There are many different types of content on the portal, such as components, iScripts, homepages, and so on. Sometimes you want to access this content directly, through the URL that points to this content. To do this, PeopleSoft provides several Generate functions that generate the URL for the specified content.

The Generate functions can be categorized as:

- Content type.
- Absolute or relative URL.
- Portal service or content servlet (psp or psc).

The following is a list of the functions, plus a description of what to use them for. More information about a function is found under that function's description in the section PeopleCode Built-in Functions.

Function Name	***Content Type***	***Absolute or Relative URL***	***Portal service or content servlet***
GenerateActGuideContentUrl	Activity Guide	Absolute	psc
GenerateActGuidePortalUrl	Activity Guide	Absolute	psp

Function Name	***Content Type***	***Absolute or Relative URL***	***Portal service or content servlet***
GenerateActGuideRelativeUrl	Activity Guide	Relative	NA
GenerateComponentContentURL	Component	Absolute	psc
GenerateComponentContentRelURL	Component	Relative	psc
GenerateComponentPortalURL	Component	Absolute	psp
GenerateComponentPortalRelURL	Component	Relative	NA
GenerateComponentRelativeURL	Component	Relative	NA
GenerateExternalPortalURL	External	Absolute	psp
GenerateExternalRelativeURL	External	Relative	NA
GenerateHomepagePortalURL	Homepage	Absolute	psp
GenerateHomepageRelativeURL	Homepage	Relative	NA
GenerateQueryContentURL	WQuery	Absolute	psc
GenerateQueryPortalURL	Query	Absolute	psp
GenerateQueryRelativeURL	Query	Relative	NA
GenerateScriptContentURL	IScript	Absolute	psc
GenerateScriptContentRelURL	IScript	Relative	psc
GenerateScriptPortalURL	Iscript	Absolute	psp
GenerateScriptPortalRelURL	IScript	Relative	psp
GenerateScriptRelativeURL	IScript	Relative	NA

Function Name	Content Type	Absolute or Relative URL	Portal service or content servlet
GenerateWorklistPortalURL	Worklist	Absolute	psp
GenerateWorklistRelativeURL	Worklist	Relative	NA

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions"

Error Handling

All errors are handled through the session object. You should check the PSMessages collection to see if there are any errors in your code.

Note.

`Exit(1)` does *not* rollback iScript transactions. To rollback in an iScript, you can use the SqlExec built-in function with the parameter of ROLLBACK (`SQLEXEC("ROLLBACK")`) or the MessageBox built-in function with a message error severity of error. You can also use the built-in function Error, but only if you are not sending HTML or XML in the error text itself.

See Also

Chapter 38, "Session Class," Error Handling, page 2180

Scope of the Internet Script Classes

The Request, Response, and Cookie classes can be accessed only from PeopleCode. They can be used only as part of a PeopleSoft Pure Internet Architecture application—either as part of a page created in Application Designer or an iScript. An iScript is always contained in a WEBLIB record. You wouldn't use any of these objects in an Application Engine program, a Component Interface, and so on. However, an iScript could call and start a Component Interface, a Business Interlink, and so on.

To access the Request or Response object, use the %Request and %Response system variable. These variables are objects, so you use them with dot notation.

```
%Response.SetContentType("text/HTML");

If %Request.GetParameter("encode") <> "" Then
```


Data Types of the iScript Classes

Cookie objects are instantiated from Response object. You can declare the cookie object as data type cookie before any functions. You can also declare variables of type Response and Request, to assign the %Response and %Request system variables to local variables.

```
Local Cookie &Cookie;
Local Response &Resp;
Local Request &Req;

Function IScript_SetCookie()
    rem set a cookie called "MyCookie" to store my user id in it's value. 
Make the cookie expire when the user's browser session expires;

   &Resp = %Response;
   &Req = %Request;
   &Cookie = &Resp.CreateCookie("MyCookie");
   &Cookie.Value = %UserId;
   &Cookie.MaxAge = -1;

End-Function;
```

Internet Script Classes

The following is a description of the PeopleCode classes, methods, and properties that you use to create an iScript.

Request Class

The Request object encapsulates all information from the request issued from the browser. This includes the URI, Query String, QueryString parameters, Cookies, and Headers. The properties for the Request object are read-only.

Parameters

Request parameters are name-value pairs sent by the client to the web server as part of an HTTP request. They include the pairs in the query string part of the URL, and data posted in a form, if the request was a post request. Multiple parameter values can exist for any given parameter name. The following methods are available to access parameters:

- GetParameter
- GetParameterNames
- GetParameterValues

The GetParameterValues method returns an array of String objects containing all the parameter values associated with a parameter name. The value returned from the GetParameter method equals the first value in the array of String objects returned by GetParameterValues.

All form data from both the query string and the post body are aggregated into the request parameter set. The order of this aggregation is that query string data appears before post body parameter data.

Response

The response object encapsulates all the information to be sent back to the browser. This includes the body of the response, content type, response headers, and cookies.

The response object also includes helper methods that retrieve HTML, Image, StyleSheet, and other objects from the database and move them to the web server, and returning strings that can be used to reference these objects in the response content.

The body of the response is created using the response object's Write and WriteLine methods. If the content type of the response is not explicitly set in the PeopleCode program, it defaults to "text/html".

Cookies

Cookies are data sent from the client to the server on every request that the client makes. Cookies are stored by the browser, either on disk or in memory, and returned to the server that originally set the cookie.

The Internet Script API makes available the name and value of each cookie that's sent with the request, through the Request object's GetCookieNames and GetCookieValue methods. The API also enables you to set new cookies (or alter existing cookies) through the Response object's CreateCookie method and the Cookie object's properties.

Internet Script Classes Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateActGuideContentUrl

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateActGuidePortalUrl

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateActGuideRelativeUrl

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentContentRelURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentContentURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentPortalRelURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentPortalURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentRelativeURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateExternalPortalURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateExternalRelativeURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateHomepagePortalURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateHomepageRelativeURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateQueryContentURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateQueryPortalURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateQueryRelativeURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptContentRelURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptContentURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptPortalRelURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptPortalURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptRelativeURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateWorklistPortalURL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateWorklistRelativeURL

Request Class Methods

In this section, we discuss the Request class methods. The methods are discussed in alphabetical order.

GetContentBody

Syntax

GetContentBody()

Description

Note. PeopleSoft Business Interlinks is a deprecated product. This method currently exists for backward compatibility only.

This method retrieves the text content of an XML request. This is part of the incoming Business Interlink functionality, which enables PeopleCode to receive an XML request and return an XML response.

Parameters

None.

Returns

A string.

Example

The following example gets the XML text content of a request, then uses that as input to create a BiDoc. Then the BiDoc methods GetDoc and GetValue are used to access the value of the skills tag.

```
Local BIDocs &rootInDoc, &postreqDoc;
Local string &blob;
Local number &ret;

&blob = %Request.GetContentBody();
/* process the incoming xml(request)- Create a BiDoc */
&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetDoc("postreq");
&ret = &postreqDoc.GetValue("skills", &skills);
```

GetCookieNames

Syntax

GetCookieNames()

Description

This method returns an array containing names of all the cookies present in this request. If there are no cookies in the request, an empty array is returned. When cookies are present, this method returns an array of strings.

GetCookieValue

Syntax

GetCookieValue*(name)*

Description

This method returns a string containing value of the cookie identified *name*. The *name* parameter takes a string value. The NPmatch between *name* and the request cookie is case-insensitive. If there is no cookie in the request matching the name, an empty string is returned.

GetHeader

Syntax

GetHeader*(name)*

Description

This method returns the value of the header requested by the string *name*. The match between *name* and the request header is case-insensitive. If the header requested does not exist, an empty string is returned.

Example

The following example gets the "Referer" header to see where the request came from:

```
&Referer = %Request.GetHeader("Referer");
```

GetHeaderNames

Syntax

GetHeaderNames()

Description

This method returns an array of Strings representing the header names for this request.

GetHelpURL

Syntax

GetHelpURL*(HelpContext)*

Description

This method returns the help context path (as a string) to the help directory. It's used to construct a URL to a specific help page.

Example

```
&HelpURLString = %Request.GetHelpURL("hc0110");
```

GetParameter

Syntax

GetParameter*(name)*

Description

This method returns the value of a specified query string parameter or posted form data parameter. The match between *name* and the request parameter is case-insensitive. If there are multiple parameter values for a single name, the value returned is the first value in the array returned by the GetParameterValues method. If the parameter has (or could have) multiple values, you should use the GetParameterValues method in your Internet scripts.

See Also

Chapter 21, "Internet Script Classes (iScript)," GetParameterValues, page 1066

GetParameterNames

Syntax

GetParameterNames()

Description

This method returns all the parameter names for this request as an array of Strings, or an empty array if there are no input parameters.

GetParameterValues

Syntax

`GetParameterValues`*`(name)`*

Description

This method returns the values of the specified parameter *(name)* as an array of Strings, or an empty array if the named parameter does not exist. The *name* parameter takes a string value.

Request Class Properties

In this section, we discuss the Request class properties. The properties are discussed in alphabetical order.

AuthTokenDomain

Description

This property returns the web server domain as a string across which the single signon authentication token is valid. Use this property as the domain of any cookie which you want to apply across the same domain as the single signon token.

The value of this property is in the format:

```
".domain.com"
```

Note that it begins with a dot "." character. You can use this format as the value of a cookie's domain property.

However, sometimes you cannot use a value that is prefixed with a dot character. For example, the javascript document.docmain property should not begin with a dot. In this case, you must strip it off manually in your code.

Note. The value of this property is the domain across which the authentication token is valid, set in the AuthTokenDomain configuration property in the configuration properties file. The value of the system variable %AuthenticationToken is the authentication token itself.

This property is read-only.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "System Variables," %AuthenticationToken

AuthType

Description

This property corresponds to the CGI variable AUTH_TYPE. This variable does not necessarily contain a value, and may require special web server configuration to obtain non-empty values.

This property is read-only.

See Also

Chapter 21, "Internet Script Classes (iScript)," RemoteUser, page 1072 and Chapter 21, "Internet Script Classes (iScript)," ServletPath, page 1074

ByPassSignOn

Description

This property returns a Boolean value, indicating what ByPassSignOn has been set to in the configuration properties file. If this value has been set to True, this property returns True, False otherwise.

This property is read-only.

BrowserPlatform

Description

This property returns a string containing the value the web server passes as the browser platform.

This property is read-only.

BrowserType

Description

This property returns a string describing the browser that sent the request.

This property is read-only.

BrowserVersion

Description

This property returns a string describing the version of the browser that sent the request.

This property is read-only.

ContentURI

Description

This property returns a string containing the portion of the current URI before the portal name, and referencing the content servlet (psc).

For the URL:

```
http://localhost/psc/ps84/PORTAL/NODE/e_procurement/fdm/s/WEBLIB. . .
```

This property returns the following:

```
http://localhost/psc/ps84/
```

This property is read-only.

ExpireMeta

Description

This property returns the refresh meta-tag string that contains the cmd=expire parameter (the ExpireMeta string.)

The following is an example of the ExpireMeta string:

```
<meta HTTP-EQUIV='Refresh' Target='_top' CONTENT='10; URL=/servlets/psp/ps84/?cmd=⇒
expire'>
```

All Internet script pages should use this property to generate the meta tag to cause the page to expire. It is the same expiration tag used by pages originally created using Application Designer, unless the Internet script generates a menu link in a separate frame that is not supposed to expire.

This tag should be included in the <head> section of the HTML generated by the iScript.

This property is read-only.

Example

```
%Response.Write("<html><head>");
%Response.Write(%Request.ExpireMeta);
%Response.Write("</head><body>");
```

FullURI

Description

This property returns the complete URI up to, but not including, the query string. This method returns a string value.

To return only the request URI (that is, without the scheme, server name or port) use the RequestURI method.

This property is read-only.

Example

From the following code

```
&FullURI = &Request.FullURI;
```

&FullURI might contain the following:

```
http://serverx/servlets/psp/ps84/Portal/Node/s/WEBLIB_TEST.SCRIPTS.Field⇒
Formula.IScript_Test
```

See Also

Chapter 21, "Internet Script Classes (iScript)," RequestURI, page 1072

HTTPMethod

Description

This property returns the HTTP method as a string, (for example, GET, POST, PUT) by which this request was made.

This property is read-only.

LogoutURL

Description

This property returns the complete URL (as a string) to logout of the PeopleSoft session. Use this property to generate a link that causes a page to logout. You should normally not need to include this link on your page.

This property is read-only.

PathInfo

Description

This property returns any path information following the servlet path, but prior to the query string. This property returns null if there is no path information following the servlet path.

This property is read-only.

Example

For the URL:

```
http:/localhost/psp/ps84/PORTAL/NODE/e_procurement/fdm/s/WEBLIB. . .
```

This property returns the following:

```
psp84/PORTAL/NODE/eprocurements/fdm/s/WEBLIB_. . .
```

Protocol

Description

This property returns the protocol being used for this request as a string in the following form:

protocol/major_version.minor_version

An HTTP 1.0 request, as defined by the HTTP 1.0 specification, should return the string HTTP/1.0. Use the Scheme property instead of the Protocol property when generating hrefs (links).

This property is read-only.

QueryString

Description

This property returns the query string present in the request URL, if any. A query string is defined as any information following a ? character in the URL.

If there is no query string, this method returns null.

Use the request Parameters methods (GetParameterNames, GetParameterValues, GetParameter) to get the values of individual parameters on the query string.

This property is read-only.

RelativeURL

Description

This property returns whether a relative URL should be generated for PeopleSoft Pure Internet Architecture pages. This property is set in the configuration properties file. This property takes a Boolean value: True if a relative URL is generated for PeopleSoft Pure Internet Architecture pages, False otherwise.

This property is read-only.

RemoteAddr

Description

This property returns the IP address of the agent that sent the request.

This property is read-only.

RemoteHost

Description

This property returns the fully qualified host name of the agent that sent the request.

This property is read-only.

RequestURI

Description

This property returns the URI, without the protocol.

This property is read-only.

Example

This example uses the following URL:

```
http://serverx/servlets/psc/ps84/PORTAL/NODE/fdm/s/WEBLIB_Portal. . .
```

From the following code

```
&MyRequestURI = &Request.RequestURI;
```

&MyRequestURI contains the following:

```
/servlets/psc/ps84/PORTAL/NODE/fdm/s/WEBLIB_PORTAL . . .
```

Use the following to build an absolute URL:

```
&myURL = &Request.Scheme | "://" | &Request.ServerName | ":" | &Request.Port | ⇒
&Request.RequestURI | "?" | &Request.QueryString;
```

RemoteUser

Description

This property corresponds to the CGI variable REMOTE_USER. It is not specific to your PeopleSoft implementation.

This variable does not necessarily contain a value, and may require special web server configuration to obtain non-empty values.

This value is generally populated when J2EE Authentication is enabled in the web-server configuration. The specific implementation of your web-server dictates how exactly this variable is populated. See your web-server specific documentation for more information.

This property is read-only.

See Also

Chapter 21, "Internet Script Classes (iScript)," AuthType, page 1067 and Chapter 21, "Internet Script Classes (iScript)," ServletPath, page 1074

Scheme

Description

This property returns the scheme, also known as the protocol, used by the request. Common schemes include http, https, ftp, and telnet. This property returns a string value.

This property is read-only.

Example

For the URL

```
http://serverx/servlets/psp/ps84/eprocurement/fdm/s/WEBLIB. . . .
```

the Scheme property returns:

```
http
```

ServerName

Description

This property returns the host name of the server on which the servlet is running. This property returns a string value.

This property is read-only.

Example

For the URL

```
http://serverx/servlets/psp/ps84/eprocurement/fdm/s/WEBLIB. . . .
```

the ServerName property returns:

```
serverx
```

ServerPort

Description

This property returns an integer representing the port on which the browser's request was received (that is, the port on which the server is listening.)

This property is read-only.

Example

For the URL

`http://servername:80/servlets/iclientservlet/peoplesoft8/?ICType=Panel. . .`

the ServerPort property returns:

`80`

ServletPath

Description

This property corresponds to the CGI variable SCRIPT_NAME. This variable does not necessarily contain a value, and may require special web server configuration to obtain non-empty values.

This property is read-only.

See Also

Chapter 21, "Internet Script Classes (iScript)," AuthType, page 1067 and Chapter 21, "Internet Script Classes (iScript)," RemoteUser, page 1072

Timeout

Description

This property returns an integer representing the timeout value, in seconds, set in the configuration.properties file.

This property is read-only.

Response Class

The response object encapsulates all information to be returned from the iScript to the browser.

Response Class Methods

In this section, we discuss the Response class methods. The methods are discussed in alphabetical order.

Clear

Syntax

`Clear``()`

Description

This method removes all cookies and headers as well as deletes any HTML that has been written to the Response object. After using this method, the Response object appears as it did when the script was first called.

CreateCookie

Syntax

`CreateCookie`*`(name)`*

Description

This method adds a cookie to the response with the name specified by the string *name.* It returns a reference to the cookie object that is used to update the values of this cookie. If the cookie by the specified name already exists, a reference to the existing cookie is returned. This method can be called multiple times to set more than one cookie. Cookie names may not contain the "$" character.

Important! Certain browsers have limit of either 20 or 50 cookies per web server domain depending on the browser security update that the user has applied. The browser randomly discards some cookies when that number is exceeded.

PeopleTools uses eight cookies. In addition, the number of cookies used by PeopleTools can grow depending on the number of sites that the user visits within the same domain. Some PeopleSoft applications use additional cookies. Therefore, you might or might not be able to create your own cookies to modify an existing PeopleSoft application. If you do create your own cookies, keep the use conservative. Otherwise, your application might end up causing the browser to discard critical signon or authentication cookies.

More information about browser limitations can be found online.

See "Troubleshooting Browser Limitations" on My Oracle Support.

GetCookie

Syntax

`GetCookie`*`(name)`*

Description

This method returns the cookie object specified by the string *name*. If the cookie is not already present in this response, a null value is returned.

GetCookieNames

Syntax

GetCookieNames()

Description

This method returns an array of strings that contains the names of all the cookies present in this response. If there are no cookies in the response, an empty array is returned.

GetHeader

Syntax

GetHeader*(name)*

Description

This method returns the value of the response header requested by the string *name*. The match between *name* and the response header is case-insensitive. If the header requested does not exist, an empty string is returned.

GetHeaderNames

Syntax

GetHeaderNames()

Description

This method returns an array of Strings representing the header names for this response.

GetImageURL

Syntax

GetImageURL(IMAGE.*ImageName* | *rowset.row.record*)

Description

This method returns a string which represents the URL of the requested image, for images in the database.

Because the image field is a long data type, you can have only one on the record. Therefore you must specify the record only. The field on the record is assumed to be the image.

Note. The GetImageURL method cannot be used with a derived work record.

Parameters

Parameter	***Description***
imagename	Specify the name of an existing image. This name must be prefaced with the reserved word **IMAGE.**
rowset.row.record	Specify the path to the image, that is, the rowset, row, and record.

Example

The following example gets the URL to use for the image, and the second uses the URL reference.

```
&strImage = %Response.GetImageURL(Image.ARROW);
```

```
%Response.WriteLine("<img src=" | &strImage | ">");
```

The following gets an image from the record.

```
%Response.GetImageURL(&rs.GetRow(1).GetRecord(Record.EMPL_PHOTO));
```

GetJavaScriptURL

Syntax

GetJavaScriptURL(**HTML.***HTMLname*)

Description

This method returns a string which represents the URL of the requested HTML definition, for JavaScript HTML definitions stored in the database.

Example

This example assumes the existence in the database of an HTML definition called "HelloWorld_JS", that contains some JavaScript.

```
Function IScript_TestJavaScript()

   %Response.WriteLine("<script src= " | %Response.GetJavaScriptURL(HTML.Hello⇒
World_JS) | "></script>");

End-Function;
```

In this example, the HTML definition is called FOO. The following PeopleCode example:

```
%Response.GetJavaScriptURL(HTML.FOO)
```

Resolves to something similar to:

```
http://myserver/cache/js/FOO_ENG_1.js
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating HTML Definitions"

GetStyleSheetURL

Syntax

GetStyleSheetURL(STYLESHEET.*stylename*)

Description

This method returns a URL string pointing to a style sheet created in Application Designer.

Example

In the following example, the first line gets the URL to use for the style sheet, and the second is the standard HTML to include the style sheet in the HTML document. The stylesheet link must be included in the HTML *before* any of the styles in the stylesheet are used. It should be included in the <head> section of the HTML document.


```
&strStyleSheet = %Response.GetStyleSheetURL(StyleSheet.BENEFITS_STYLE);

%Response.WriteLine("<link rel=stylesheet href=" | &strStyleSheet | " type=""text⇒
/css"">");
```

Now to use a style, assign the class attribute of your choice to your HTML tags, as follows:

```
%Response.WriteLine("<p class=PSTEXT>I am some classy text!!</p>");
```

RedirectURL

Syntax

RedirectURL*(location)*

Description

This method sends a temporary redirect response to the client using the location specified by the string location. The given location must be an absolute URL. No further output should be made by the iScript after calling this method.

Note. If your URL string contains special characters (such as foreign characters) make sure you encode it first using the EncodeURL function, such as, `%Response.RedirectURL(EncodeUrl(&Url))`.

In addition, while you can redirect relative URLs in certain circumstances, you *cannot* encode them.

Example

```
%Response.RedirectURL(&URL);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," EncodeURL

SetContentType

Syntax

SetContentType*(Type)*

Description

This method sets the content type for this response. The parameter *type* takes a string value. This type may later be implicitly modified by the addition of properties such as the MIME charset property if the service finds it necessary and the appropriate property has not been set.

The content type defaults to "text/html" if it is not set.

SetHeader

Syntax

SetHeader*(name,value)*

Description

This method sets a response header with the specified name and value. Both parameters take string values. If the field has already been set with a value, this new value overwrites the previous one.

Example

The PortalRegisteredURL response header identifies to the portal the registered URL of the current content, so that it can find the correct content reference to look at for the template.

You can override the value of the PortalRegisteredURL response header in a PIA Script or PIA Page by adding the header to the response yourself, like this:

```
%Response.SetHeader("PortalRegisteredURL", &myURL);
```

You can do this to register the content with additional parameters.

UseSimpleURL

Syntax

UseSimpleURL({True | False})

Description

The UseSimpleURL method sets the response header that informs the portal that the current content uses relative simple URLs, and does not have to be proxied.

By default, all PeopleSoft Pure Internet Architecture content sets the header to true. You must to call this method to specify that your content does not use the simple URL format, and therefore does need to be proxied.

Parameters

Parameter	*Description*
True | False	Specify a Boolean value, either True or False, to indicate whether simple URLs are used.

Returns

None.

Write

Syntax

`Write`*`(String)`*

Description

This method prints *String* to the HTTP output stream.

You can use an HTML string from Application Designer HTML catalog with the Write method if you also use the GetHTMLText function, as follows:

```
%Response.Write(GetHTMLTEXT(HTML.MY_HTML));
```

You can also use an XML string. The following example takes a BiDocs structure that contains an XML response and puts that into a text string. After this is done, the %Response.Write function can send this as an XML response.

```
Local BIDocs &rootDoc;
Local string &xmlString;

&xmlString = %Response.GetContentBody();
&rootDoc = GetBIDoc(&xmlString);

/* do processing */

&xmlString = &rootDoc.GenXMLString();
%Response.Write(&xmlString);
```

WriteLine

Syntax

`WriteLine`*`(String)`*

Description

This method adds a carriage control and line feed to the end of the string *String*, then prints the string to the HTTP output stream.

You can use an HTML string from Application Designer HTML catalog with the WriteLine method if you also use the GetHTMLText function, as follows:

```
%Response.WriteLine(GetHTMLTEXT(HTML.MY_HTML));
```

Response Class Properties

In this section, we discuss the Response class properties.

Charset

Description

This property returns the character set used by the response object as a string. This value is set in the user personalizations.

This property is read-only.

See Also

PeopleTools 8.51 PeopleBook: Security Administration, "Managing PeopleSoft Personalizations"

DefaultStyleSheetName

Description

This property returns the name of the default style sheet defined on the PeopleTools Options page. If not defined, PSSTYLEDEF is returned.

This property is read-only.

See Also

PeopleTools 8.51 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," PeopleTools Options

Cookie Class

The cookie class encapsulates all information to be sent to the browser for a single cookie. The cookie class is used by the response object to set new cookies. The request object uses a simple name-value pair mechanism for retrieving previously stored cookie values sent with the HTTP request.

Cookie Class Properties

In this section, we discuss the Cookie class properties. The properties are discussed in alphabetical order.

Domain

Description

This property returns the domain of this cookie, or null if not defined. This property is a string value.

This property is read-write.

Example

```
&cookie = &Response.AddCookie("My cookie", "My value");

&cookie.Domain = ".MyDomain.com";
```

MaxAge

Description

This property represents the maximum specified age of the cookie, as a signed number in seconds. The default value is -1. Setting the MaxAge property to a negative value ensures the cookie does not persist on the client when the client session ends. If the MaxAge property is set to zero, the cookie is deleted immediately from the client.

Value	***Description***
Non Negative Integer	Lifetime of the Cookie in seconds
0	Delete the cookie from the client immediately
-1	Default MaxAge property value (remove the cookie after the client exits)
Negative Integer	Remove the cookie after the client exits

This property is read-write.

Name

Description

This property returns the name of the cookie as a string.

This property is read-only.

Path

Description

This property returns the prefix of all the URL paths for which this cookie is valid, or an empty string if not defined.

This property is read-write.

Secure

Description

This property specifies whether the cookie is to be secured. This property takes a Boolean value. The default value is False. Secure cookies are sent only if the client has established a secure link (such as HTTPS) with the server. This property is read-write.

Value

Description

This property returns the value of the cookie (as a string), or an empty string if it isn't defined.

This property is read-write.

Chapter 22

Java Class

This chapter provides an overview of Java class and discusses the following topics:

- Supported versions of Java.
- Naming standards for classes and packages.
- System setup for Java classes.
- From PeopleCode to Java.
- From Java to PeopleCode.
- PeopleCode and Java data types mapping.
- Considerations when using the PeopleCode Java functions.
- Error handling and the PeopleCode Java functions.
- The Java debugging environment.
- Java object declaration.
- Scope of a Java object.
- PeopleCode Java built-in functions.

Understanding Java Class

Using the PeopleCode Java functions, you can access Java classes, or create instances of Java objects. Calling a Java program from PeopleCode can greatly extend your application. Also, the PeopleCode integration with Java enables you to pass parameters such as record, record fields, or rowsets into Java programs.

The PeopleCode Java functions are:

- CopyToJavaArray
- CopyFromJavaArray
- CreateJavaArray
- CreateJavaObject
- GetJavaClass

See Also

http://java.sun.com/index.html

Supported Versions of Java

The supported platform database lists all supported versions of the Java Runtime Environment (JRE) for each version of PeopleTools on each supported operating system platform.

See My Oracle Support, "Platform Certifications."

Oracle also bundles a supported JRE version with PeopleTools for all supported operating system platforms except the z/OS platform. For z/OS, you will have to install a supported JRE. To determine which JRE is bundled with your version of PeopleTools, see the README file in the *PS_HOME*\jre directory.

Java Packages and Classes Delivered with PeopleTools

Java classes delivered with PeopleTools are located in the following directory:

`PS_HOME\class`

For PeopleSoft software, we typically use a standard package hierarchy:

`com.peoplesoft.prodline.prodcode`

In this hierarchy, *prodline* is the company standardized product line code (for example, hrms) and *prodcode* is the product code unique within the product line (for example, hr).

In addition, if there are any classes that are common across products, they are placed into the common package for that product line, that is:

`com.peoplesoft.prodline.common`

See Also

http://java.sun.com/docs/codeconv/index.html

System Setup for Java Classes

If you access only the classes that come defined with PeopleTools, you don't need to do any additional setup.

If you want to access third-party Java classes or your own custom Java classes in the PeopleTools environment (that is, access these classes through PeopleCode), you must place the class files in specified locations or include the class files in JAR files in specified locations.

Note. Oracle recommends using the utility that comes with the Java SDK for creating JAR files.

When PeopleTools loads the Java Virtual Machine (JVM), PeopleTools builds a class path for the JVM from the following elements. The following numbering indicates the search order that the JVM would use for locating a specified Java class:

1. Class files in the *PS_HOME*\class directory.
2. Class files in JAR files in the *PS_HOME*\class directory.
3. Class files in the *PS_HOME*\appserv\classes directory.
4. Class files in JAR files in the *PS_HOME*\appserv\classes directory.
5. For each directory listed in the "Add To CLASSPATH" parameter of the psappsrv.cfg configuration file:
   a. Class files in the specified directory.
   b. Class files in JAR files in the specified directory.
6. For each directory listed in the PS_CLASSPATH environment variable:
   a. Class files in the specified directory.
   b. Class files in JAR files in the specified directory.

For example, if PS_CLASSPATH is "dir1;dir2", the search order for item 6 described previously would be:

1. Class files in dir1.
2. Class files in JAR files in dir1.
3. Class files in dir2.
4. Class files in JAR files in dir2.

Note. PeopleTools uses the -classpath option when it loads the JVM, which overrides the CLASSPATH environment variable. Therefore, do not use the CLASSPATH environment variable to identify third-party or custom Java classes that you want to access in the PeopleTools environment.

Note. PeopleTools does not guarantee the order in which JAR files within a directory will be added to the class path. If it is necessary to impose a search order on JAR files, then the JAR files must be in separate directories.

Like most environment variables, you can specify more than one entry in PS_CLASSPATH. On Windows, the PS_CLASSPATH entries are separated by semicolons. On Unix, they're separated by colons.

The following PS_CLASSPATH is for Windows:

```
c:\myjava;d:\myjava\com\mycompany\myproduct; . . .
```

The following PS_CLASSPATH would be for Unix:

```
/etc/myjava:/home/me/myjava/com/mycompany/myproduct: . . .
```


When developing your own classes you must be aware that most JVMs cache the class definitions. This means that even if you update the class files, a running JVM (inside an application server, for example) has already loaded and is referencing the old versions of the class files. The JVM won't pick up the new versions of the class files. You must restart the application server to make the JVM reload the updated classes.

See your system documentation for more information about setting an environment variable.

From PeopleCode to Java

In this section, we provide an overview of state management concerns and present the following examples:

- State Management Concerns.
- CreateJavaObject example.
- CreateJavaArray example.
- GetJavaClass example.

State Management Concerns

The application server is *stateless,* which means that it doesn't keep any information (state) for its clients between calls to it. For one reason, calls to the application server can use different actual servers for different calls. When you are using Java in the application server, be careful to not leave state in the JVM that would cause your application to fail if a different application server (which would use a different invocation of the JVM) was used for subsequent calls. One method to leave state in the virtual machine is to use static (class) variables.

Similar considerations to these apply using Java in Application Engine programs, though here the difficulty arises when you try to checkpoint and then restart the program. The restart starts with a JVM invocation that doesn't have any of the state you might have stored into the JVM before the checkpoint.

Variables of the type JavaObject cannot have global or component scope because of this lack of ability to save the state of these objects.

An example of this is issuing messages. When you're running with PeopleSoft Pure Internet Architecture and issue a message, the message is produced by an end-user action, so the Application Server gathers up its state to return it to the browser. This state saving attempts to save the current PeopleCode execution state, causing it to issue an error because of the JavaObject.

The solution is to not have any non-null JavaObject objects when the message is issued.

The following is a simple Java program:


```
public class PC_Java_Test{

public String pcTest(){

   String message;

   message = "PeopleCode is successfully executing Java.";

   return message;
   }
}
```

Here is the PeopleCode that calls this Java program. Note that the JavaObject is set to NULL before the message is issued.

```
&java_test = CreateJavaObject("PC_Java_Test");
&java_message = &java_test.pcTest();
&java_test = Null;
WinMessage(&java_message);
```

In SavePreChange, Workflow, or SavePostChange PeopleCode the situation is more complicated. Usually messages with a zero style parameter (no buttons other than OK and perhaps Explain, therefore no result possible except OK) are queued up by the Application Server. They are output by the browser when the service completes, so the serialization won't happen until after the PeopleCode has finished, so you won't have to set your JavaObject to null. With other kinds of messages, you must do this.

CreateJavaObject Example

The following is an example program creating a Java object from a sample program that generates a random password.

```
/* Example to return Random Passwords from a Java class */
Local JavaObject &oGpw;

/* Create an instance of the object */
&oGpw = CreateJavaObject("com.PeopleSoft.Random.Gpw_Demo");

&Q = "1";

/* Call the method within the class */
&NEW_VALUE = &oGpw.getNewPassword(&Q, PSRNDMPSWD.LENGTH);

/* This is just returning one value for now */
PSRNDMPSWD.PSWD = &NEW_VALUE;
```

CreateJavaArray Example

Suppose we had a PeopleCode array of strings (&Parms) that we wanted to pass to a Java method xyz of class Abc. This example assumes that you don't know when you write the code just how many parameters you will have.


```
Local JavaObject &Abc, &RefArray;
Local array of String &Parms;

&Parms = CreateArray();

/* Populate array how ever you want to populate it */

&Abc = GetJavaObject("com.peoplesoft.def.Abc");

/* Create the java array object. */

&JavaParms = CreateJavaArray("java.lang.String[]", &Parms.Len);

/* Populate the java array from the PeopleCode array. */

&RefArray = GetJavaClass("java.lang.reflect.Array");

For &I = 1 to &Parms.Len
   &RefArray.set(&JavaParms, &I - 1, &Parms[&I]);
End-For;

/* Call the method. */
&Abc.xyz(&JavaParms);
```

GetJavaClass Example

The following example gets a system class.

```
&Sys = GetJavaClass("java.lang.System");
&Sys.setProperty("java.security.policy", "C:\java\policy");

WinMessage("The security property is: " | &Sys.getProperty("java.security.policy"));

&Props = &Sys.getProperties();
&Props.put("java.security.policy", "C:\java\policy");
&Sys.setProperties(&Props);

WinMessage("The security property is: " | &Sys.getProperty("java.security.policy"));
```

From Java to PeopleCode

The Java classes delivered with PeopleTools enable you to call PeopleCode from your Java program. Calling into PeopleCode works only from Java code that you have initially called from PeopleCode.

You must call PeopleCode facilities only from the same thread that was used for the call into Java. PeopleTools is not multithreaded.

You cannot call any PeopleCode facility that would cause the server to return to the browser for an end-user action, because the state of the Java computation cannot be saved and restored when the action is complete.

See Also

Chapter 22, "Java Class," Considerations When Using the PeopleCode Java Functions, page 1096

SysVar Java Class

Use the SysVar Java Class to refer to System Variables, such as %Language or %DBType.

For example, %Session, becomes SysVar.Session()

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "System Variables"

SysCon Java Class

Use the SysCon Java Class to refer to system constants, such as %SQLStatus_OK or %FilePath_Absolute.

For example, %CharType_Matched becomes SysCon.CharType_Matched.

Func Java Class

Use the Func Java Class to refer to built-in functions, such as CreateRowset or GetFile.

For example, SetLanguage(LANG_CD) becomes Func.SetLanguage(LANG_CD)

Name Java Class

The Name Java Class enables you to use the PeopleSoft reserved item references. This enables you to reference pages, components, records, fieldnames, and so on.

For example, in PeopleCode you can refer to a record field using the following:

```
recname.fieldname
```

With the Name class, you can use a similar construct:

```
new PeopleSoft.PeopleCode.Name("RECNAME", "FIELDNAME");
```

Note that these must be in the exact case as the item. As all PeopleTools items are named in uppercase, that means you must use uppercase.

As another example, in PeopleCode you can refer to a page using the following:

```
PAGE.pagename
```

In Java, it would be:

```
new PeopleSoft.PeopleCode.Name("PAGE", "PAGENAME");
```

Accessing PeopleCode Objects

The existing PeopleCode classes (like Array, Rowset, and so on) have properties and methods you can access.

- PeopleCode classes have the same names, so Record becomes Record, SQL becomes SQL, and so on.
- Methods are accessed by the method name.
- The name of a property is pre-pended with either **get** or **set,** depending on whether you're reading or writing to the property.

  For example, to get the IsChanged property would be getIsChanged. To set the value for a field would be &MyField.setValue.

Here is an example of a Java program that uses PeopleCode objects to access the database:


```
/*
 * Class Test
 *
 * This code is used to test the Java/PeopleCode interface.
 *
 */

import PeopleSoft.PeopleCode.*;

public class Test {

/*
 * Test
 *
 * Add up and return the length of all the
 * item labels on the UTILITIES menu,
 * found two different ways.
 *
 */

public static int Test() {
/* Get a Rowset to hold all the menu item records. */
Rowset rs = Func.CreateRowset(new Name("RECORD", "PSMENUITEM"), new Object[]{});
String menuName = "UTILITIES";
int nRecs = rs.Fill(new Object[]{"WHERE FILL.MENUNAME = :1", menuName});

int i;
int nFillChars = 0;
for (i = 1; i <= rs.getActiveRowCount(); i++) {
String itemLabel = (String)rs.GetRow(i)
.GetRecord(new Name("RECORD", "PSMENUITEM"))
.GetField(new Name("FIELD", "ITEMLABEL"))
.getValue();
nFillChars += itemLabel.length();
}

/* Do this a different way - use the SQL object to read each menu
   item record. */

int nSQLChars = 0;
Record menuRec = Func.CreateRecord(new Name("RECORD", "PSMENUITEM"));
SQL menuSQL = Func.CreateSQL("%SelectAll(:1) WHERE MENUNAME = :2",
new Object[]{menuRec, menuName});

while (menuSQL.Fetch(new Object[]{menuRec})) {
String itemLabel = (String)menuRec
.GetField(new Name("FIELD", "ITEMLABEL"))
.getValue();
nSQLChars += itemLabel.length();
}

return nFillChars + 100000 * nSQLChars;
}
}

This can be run from PeopleCode like this:
Local JavaObject &Test;
Local number &chars;

&Test = GetJavaClass("Test");
&chars = &Test.Test();

&Test = Null;
WinMessage("The character counts found are: " | &chars, 0);
```


Using Application Classes From Java to PeopleCode

You call a Java program from an Application Class the same way you do using any other PeopleCode program, that is, by using one of the existing Java class built-in functions.

Calling an Application Class from a Java program has the following considerations:

- Application Classes must be accessed using the object built-in functions, such as CreateObject, ObjectDoMethod, ObjectGetProperty, and so on.
- You cannot declare a variable of type HR.Package.SomeClass in your Java program. The variable must be of type Object.
- There is no pre-pending the word 'get' or 'set' for properties. All classes, methods, and properties are passed as strings.

The following is an example of how to call an Application Class from a Java program.

This is the Java program:

```
package com.peoplesoft.pcode;
import PeopleSoft.PeopleCode.*;

public class foo {

  public foo() {
  }
  public String getString() {

    Object foo = Func.CreateObject("GTP:Foo", new Object[]{});
    return (String)Func.ObjectDoMethod((Peer)foo, "GetString", new Object[]{});
  }
}
```

The following is the Application Class Foo, in the Application Package Foo:

```
class Foo
   method GetString() Returns string;
end-class;

method GetString
   /+ Returns String +/
   Return "Hello";
end-method;
```

The following is the PeopleCode program that starts it all:

```
Local JavaObject &foo = CreateJavaObject("com.peoplesoft.pcode.foo");
GTP_PARSER.GTP_STR_RESULT = &foo.getString();
```

PeopleCode and Java Data Types Mapping

The following table describes the matching of types for resolution of overloaded Java methods and basic conversions. The first Java Type/Class is the one that is produced in the absence of any other type of information.

PeopleCode Type	***Java Type/Class***
Float	double, float
Number	double, float, byte, char, short, int, long
Integer	int, byte, char, short, long
Boolean	Boolean
String	java.lang.String
Date	java.sql.Date
Time	java.sql.Time
Date Time	java.util.Date
any kind of object	any kind of object

The following table represents the conversions done to produce the Java class java.lang.Object. In addition to these, the conversions (listed in the previous table) from String onwards are done to produce a java.lang.Object.

PeopleCode Type	***Java Type/Class***
Float, Number	java.lang.Double
Integer	java.lang.Integer
Boolean	java.lang.Boolean

The following table represents other conversions that are done as required by the signature of a Java method or constructor.

PeopleCode Type	***Java Class***
Integer, Number, Float	java.lang.Integer, java.lang.Byte, java.lang.Character, java.lang.Short, java.lang.Long, java.lang.Float, PeopleSoft.PeopleCode.intHolder, PeopleSoft.PeopleCode.doubleHolder

PeopleCode Type	***Java Class***
String	PeopleSoft.PeopleCode.StringHolder
peoplecode builtin class Xxx	PeopleSoft.PeopleCode.Xxx
JavaObject	corresponding Java object

Considerations When Using the PeopleCode Java Functions

Some PeopleCode built-in functions can't be called from a Java program. Many of these restrictions arise because you can't serialize Java objects. Inside Java, you can't serialize and save the state of the JVM.

This means that you cannot call the following built-in functions:

- Think-time functions, such as DoCancel, Prompt, RevalidatePassword, and so on.
- Math functions, such as Abs, Cos, and Sin. Use the Java math functions instead.
- Java class functions, such as CreateJavaArray, CreateJavaObject, or GetJavaClass.
- Deprecated functions, such as SetNextPanel, PanelGroupChanged, and so on. Use the new function instead, like SetNextPage, ComponentChanged, and so on.
- PeopleCode language elements, such as Exit, Return, If, and so on.
- Cursor position, such as SetCursorPos.
- Functions that rely on specific character encoding, such as char, code, exact, codeb, and so on.
- Menu appearance functions, such as CheckMenuItem, HideMenuItem, and so on.
- Transfer functions such as Transfer or TransferPage.
- DoSaveNow isn't allowed. However, DoSave is.

When you're creating your Java program, keep the following points in mind:

- If you're starting from an online application, avoid calling anything that will wait a long time.
- If you use third-party Java that require third-party platforms, you are limiting your program to just those platforms.

When setting a null date in Java, use the following:

```
myField.SetValue("");
```


See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Think-Time Functions

Copying Arrays of Data Between PeopleCode and Java

When PeopleCode is called from your Java program it executes what are called native methods. These methods looks like regular Java methods in their definition but are implemented in the PeopleTools layer. In order to go from Java into PeopleTools you have to use an interface called the Java Native Interface (JNI). There is a cost associated with each transition across the JNI. Using the CopyToJavaArray and the CopyFromJavaArray functions may improve your performance, as they act as a type of bulk data copy that minimizes the transition overhead.

For example you could copy an array by copying each element using the array's Get method. However, that would require traversing the JNI twice for each element; once going into Tools and once coming back from Tools. Not only is there transition overhead but there is also conversion between object types. For example a PeopleCode string has to be converted to a Java String and vice-versa. While these two builtins do not eliminate the latter conversions, they minimize the number of transitions across the JNI.

These functions can be used when you are selecting data into a PeopleCode array and you want to copy that data into a Java array.

These functions also allow you to supply an optional parameter that specifies the list of items you want copied.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CopyFromJavaArray

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CopyToJavaArray

Considerations Working with the Java Garbage Collector

Memory is managed in the JRE by the Java garbage collector. Because PeopleTools has no control over when the Java garbage collector runs (apart from a user in their Java program calling it using `system.gc()`), PeopleTools interacts dynamically with the Java runtime garbage collector. In particular, PeopleCode objects created from a Java program (such as records, fields, and so on,) are in effect peer objects of real PeopleCode objects in PeopleTools. The release of these objects must be linked. Using the weak reference mechanism in Java, PeopleTools dynamically interracts with the Java garbage collector each time the execution thread passes through the JNI. This allows long-running Java applications in a PeopleTools context to function without having to wait for an "end of service" event for object cleanup.

There might be occasions where a Java application creates many application classes and wants to force the PeopleTools garbage collector to run. That can be achieved by calling the CollectGarbage function. This is not normally necessary.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CollectGarbage

Error Handling and the PeopleCode Java Functions

Java functions throw exceptions to indicate something unusual has happened. However, all exceptions from Java called by PeopleCode are turned into fatal errors. You can catch these exception by enclosing the call to Java in a Try-Catch PeopleCode block. If you do not try to catch these exception, the PeopleCode program is terminated, and the user transaction must be canceled.

PeopleSoft recommends that you write a Java wrapper to handle errors.

Use either the All or None built-in functions to check values that are returned if you think you may call a Java method that is defined to return a string, but returns a Null object reference instead. Java Null object references are automatically converted into PeopleCode Null object references.

Accessing the Application Log File

For doing additional error handling in your application, you can access the application log file using the PeopleCode WriteToLog built-in function. For example:

```
Func.WriteToLog(SysCon.ApplicationLogFence_Level1, myLogString);
```

See *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," WriteToLog.

The Java Debugging Environment

To use Sun's JPDA V1.0 debugging architecture, you must do the following. These instructions are general: how you actually set up the debugger depends on your system.

To use the Java Debugging Environment (JDB):

1. Download and install a copy of JPDA V1.0.
2. Set the path for your system.

   Suppose you install JPDA V1.0 in C:\jpda. Set your PATH environment variable to include C:\jpda\bin.
3. Set the path for the application server.

   For the application server, set the Domain Settings/Add to PATH to include C:\jpda\bin.

4. Set the JavaVM Options.

   Set the JavaVM Options to be something like the following (see the JPDA documentation for a more complete example):

   ```
   -Xdebug -Djava.compiler=NONE -Xnoagent -Xrunjdwp:transport=dt_socket,suspend=⇒
   n,address=8765,server=y
   ```

5. Run the debugger.

   After starting the tools session and causing it to start the JVM, you can use the JDB command line debugger that comes with JPDA, using a command like the following:

   ```
   jdb -connect com.sun.jdi.SocketAttach:port=8765
   ```

   You can also use the (no cost) Forte for Java Community Edition IDE from Sun or any of the Java IDEs noted on the JPDA pages.

See Also

http://java.sun.com/products/jpda

http://www.sun.com/forte/ffj/ce/index.html

Data Type of a Java Object

You should declare a Java object as type JavaObject. For example:

```
Local JavaObject &MyJavaClass;
```

Note. Java objects can be declared as type Local only.

Scope of a Java Object

A Java object can be instantiated from PeopleCode only. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

PeopleCode Java Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CopyFromJavaArray

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CopyToJavaArray

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateJavaArray

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateJavaObject

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetJavaClass

Chapter 23

Mail Classes

This chapter provides an overview of PeopleSoft MultiChannel Framework mail classes and discusses how to:

- Import mail classes.
- Create a mail object.
- Use the mail classes.
- Retrieve email from a mail server with the MCFGetMail class.
- MCFMail reference.

Understanding PeopleSoft MultiChannel Framework Mail Classes

This section discusses:

- PeopleSoft MultiChannel Framework mail classes.
- Scope of the mail classes.
- Data types of the mail classes.

PeopleSoft MultiChannel Framework Mail Classes

The PT_MCF_MAIL application package contains the following classes:

- MCFBodyPart
- MCFEmail
- MCFGetMail
- MCFHeaders
- MCFInboundEmail
- MCFMailStore
- MCFMultiPart
- MCFMailUtil

- MCFOutboundEmail
- MCFPart
- SMTPSession

Use the mail classes to develop applications to create and send emails, fetch and delete emails on the mail server, manage emails in your PeopleSoft database, and access attachments . By using the mail classes, you are not required to know the details of the GETMAILTARGET connector or the database schema.

Scope of the Mail Classes

The mail classes can be instantiated only from PeopleCode.

The mail classes can be called from a component, an internet script, or a Application Engine program.

Mail classes can be of local, global, or component scope.

Data Types of the Mail Classes

Every mail class is its own data type; that is, get mail objects are declared as data type MCFGetMail, store mail objects are declared as type MCFMailStore, and so on.

The following are the data types of the mail classes:

- MCFBodyPart
- MCFEmail
- MCFGetMail
- MCFHeaders
- MCFInboundEmail
- MCFMailStore
- MCFMailUtil
- MCFMultiPart
- MCFOutboundEmail
- MCFPart
- SMTPSession

Importing Mail Classes

The mail classes are *not* built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or a particular application class.

The application package PT_MCF_MAIL contains the following classes:

- MCFBodyPart
- MCFEmail
- MCFGetMail
- MCFHeaders
- MCFInboundEmail
- MCFMailStore
- MCFMailUtil
- MCFMultiPart
- MCFOutboundEmail
- MCFPart
- SMTPSession

To import all the mail classes, use the following import statement:

```
import PT_MCF_MAIL:*;
```

Using an asterisk after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available. The PT_MCF_MAIL application package contains no subpackages.

See Also

Chapter 6, "Application Classes," page 189

Creating a Mail Object

After you've imported the mail classes, you instantiate an object of one of those classes using the constructor for the class and the Create function.

The following example creates a new instance of the MCFGetMail class, as the variable &gm, with local scope:

```
Local MCFGetMail &gm = create MCFGetMail();
```

Using the Mail Classes

This section discusses how to use the mail classes, and includes the following:

- General lifecycle of an outbound email
- Delivery status notifications and return receipts
- Priority
- MCFBodyPart and MCFEmail considerations

General Lifecycle of an Outbound Email

The following are the general steps used to create an outbound email:

1. Create an MCFOutboundEmail object
2. Set the usual properties for an email, such as the From, Subject, Recipients, and Text properties.
3. If necessary, set the parts of the email using MCFMultipart and MCFBodyPart classes.
4. Use the Send method to send the email.

The MCFPart and MCFEmail classes provide the common functionality for the MCFBodyPart and MCFOutboundEmail (and MCFInboundEmail) classes, respectively. In general, you never need to use the MCFPart or MCFEmail classes directly in your PeopleCode programs. However, in rare cases you may use these classes if you want to extend the application package.

The SMTPSession class encapsulates all the session parameters that can be set in a Simple Mail Transfer Protocol (SMTP) session. Each MCFOutboundEmail object points to one SMTPSession object. In most cases, the parameters for the SMTP session are set from values found in the application server configuration file (psappsrv.cfg). However, you can also set these parameters by creating an SMTPSession object, setting the desired parameters, then creating an MCFOutboundEmail object.

For convenience, all SMTP attributes are also directly accessible from the MCFOutboundEmail class, so you don't need to create an SMTPSession object when you're sending just one email.

When you want to send many emails in a single SMTP session, you don't have to set the SMTP session parameter in each and every email. For this case, you can create a single SMTPSession, using either the default settings or explicitly setting the desired parameters, then use the CreateOutboundEmail method of SMTPSession object to create all the MCFOutboundEmail objects. This way all the MCFOutboundEmail objects use the same SMTP session parameters. After specifying the value of all the parameters for all the MCFOutboundEmail objects, use the SendAll method of SMTPSession object to send all the emails in one SMTP session.

All the application classes mentioned above hold all the data until one of the Send methods is called. The Send methods connect to the mail server using a JavaMail API that sends the mail.

Since all communication to the mail server happens when a Send method is called, errors are only reported when the Send method returns. The return value indicates the success or the failure of the send process. In case of failure, you can query the MCFOutboundEmail application class for the details of the error. The error details are available in properties such as MessageSetNumber, MessageNumber, ErrorDescription, ErrorDetails, and so on.

Delivery Status Notification and Return Receipts

There is an SMTP extension that supports Delivery Status Notifications (DSN) defined in RFC 1891. This may or may not be supported by the user's Mail Transport Agent (MTA). The acknowledgements do not indicate whether the user reads the message, they only acknowledge receipt of the message by their mail server. It is possible that the mail server throws the message away as spam and the end user never actually receives the email. To get such notifications, set the StatusNotifyOptions and StatusNotifyReturn properties, as in the following example:

```
&email.StatusNotifyOptions  = "SUCCESS,FAILURE";
&email.StatusNotifyReturn = "HDRS";
```

You can use the IsReturnReceiptReqd property to specify if a return receipt should be sent when the mail server receives and opens the email. This property only works if the underlying SMTP server supports this feature. The following is an example of using this property:

```
&email.IsReturnReceiptReqd = True;
```

Priority

There is no SMTP standard for message priority. Most email applications use an X−Priority value of 3 for messages with "normal" priority. More important messages have lower values and less important messages have higher values. In most cases, urgent messages have an X−Priority value of 1. The priority code used with the Mail classes, and set in the headers are as follows:

1. Highest Priority
2. High Priority
3. Normal
4. Low Priority
5. Lowest Priority

The default value is 3, that is, normal priority.

Different email programs render these different values in a variety of ways, usually using some kind of colored symbols or arrows.

Set the priority for an email using the Priority property. The following is an example:

```
&Email.Priority = 2;
```

MCFBodyPart and MCFEmail Considerations

Both MCFBodyPart and MCFEmail objects are used for both inbound and outbound email. The following sections specify which methods and properties are generally used for inbound, and which are used for outbound, for the different objects.

Inbound and Outbound MCFBodyPart Class Methods

All of the methods except the GetUnparsedHeaders method for the MCFBodyPart class are used with outbound email. The following methods are generally used with inbound email as well:

- GetHeader
- GetHeaderNames
- GetUnparsedHeaders

Inbound and Outbound MCFBodyPart Class Properties

The following properties are primarily used with inbound email. Note that some of these properties are also used with outbound email.

- AttachmentURL
- ContentType
- Filename
- IsAttachment
- Text

The following properties are primarily used with outbound email. Note that some of these properties are used with inbound email as well.

- ContentLanguage
- ContentType
- Description
- Disposition
- FileName
- FilePath
- FilePathType
- Multipart
- Text

Inbound and Outbound MCFEmail Class Properties

The following properties are used with inbound email. Note that most of these properties are also used with outbound email.

- From
- RefIDs

- ReplyIDs
- ReplyTo
- Sender
- Subject

The following properties are used with outbound email. Note that many of these properties are also used with inbound email.

- BCC
- BounceTo
- CC
- From
- Importance
- Priority
- Recipients
- RefIDs
- ReplyIDs
- ReplyTo
- Sender
- Sensitivity
- Subject

See Also

Chapter 23, "Mail Classes," MCFBodyPart Class Methods, page 1116

Chapter 23, "Mail Classes," MCFBodyPart Class Properties, page 1123

Chapter 23, "Mail Classes," MCFEmail Class Properties, page 1127

Retrieving Email From a Mail Server With the MCFGetMail Class

Use the MCFGetMail class to retrieve email from your mail server and load it in a rowset. After you have decided that an email needs to be saved to the PeopleSoft database, use MCFMailStore class.

This section covers the following topics:

- Rowset structure.

- Error messages common to all MCFGetMail methods.

Structure of Rowset Used for Email Information

The following list describes the different levels of the rowset that contains email information, and what each level contains.

Level 0

The first row at level zero is empty except for the number of rows contained within level zero and return status. The first row contains the return status for the entire batch of emails. Apart from the return status of the whole batch, each individual row has its own EMAIL_RET_STATUS. When a fatal error occurs, such as a connector size overflow, an IB overflow, or an error accessing attachments , the return status is copied to the first row. Subtle errors like unsupported encoding for a part of a individual email appear in the return status of that email only. Each row after the first row contains information about an individual email, such as the header information, the time received, and so on.

Level 1

Level one is where the parts of an email reside. The level one child rowsets are linked to the parent rowset in level zero by the MCF_EMAIL_ID, a unique ID assigned by PeopleSoft.

Note. Multipurpose Internet Mail Extensions (MIME) is an Internet standard for representing multipart and multimedia data in email. It enables email to be exchanged between different email systems. The parts may be nested. PeopleSoft embeds the Sun JavaMail API to implement support for the MIME standard. However, all parts of an email are represented in PeopleSoft as level one rowsets, regardless of the hierarchy that existed in the original email.

The following table describes the fields within the rowset at level zero.

Field	***Description***
MCF_ATTACH_LIST	Contains a list of all the file names that are attachments for a particular email. Multiple items are separated by a semicolon (;).
MCF_IS_ATT_URL	This is a Boolean value indicating whether the email body is located in the MCF_EMAIL_TEXT or in the attachment directory. If this value is 1 (true), the email body is located in the attachment directory and can be accessed using the MCF_ATT_URL value. If this value is 0 (false), then the email body is located in MCF_EMAIL_TEXT.

Field	**Description**
MCF_ATT_URL	This URL enables you to view an attachment. This URL is viewable only if the corresponding email has been authorized for the current user. This value is a relative URL. The URL becomes fully qualified only after the rowset has been saved to the PeopleSoft database and then later retrieved using the MCFMailStore class. The following is a sample URL: http://sundance.peoplesoft.com/PSAttachServlet/ps/mcfdev1/39297_31030554591_1.gif?contentType=image/gif *mcfdev1* is the mail user name. *39297__31030554591_1.gif* is the file name in the attachment directory. *contentType* is used by the browser to open a viewer associated with this type. The filename of the downloaded attachment is constructed by using the email's unique ID and the part number. ***Warning!*** Downloading the same emails twice from the mail server overwrites the associated files in the directory. If the same email is stored twice in the PeopleSoft database, the system creates two separate rows. If you delete either one of these rows, the system deletes the associated files in the directory. To avoid this situation, delete emails from the mail server after you have retrieved them. See Chapter 23, "Mail Classes," AuthorizeEmailAttach, page 1157.
MCF_IBNODE_NAME	Identifies the name of the Integration Broker node that received the email, such as MCF_GetMail. The system uses this value to construct the fully qualified URL required for viewing attachments.
MCF_ATTACH_SIZES	Indicates the size of the attachments associated with an email. Multiple values are separated by a semicolon (;). The attachment sizes appear in the same order as the file names in MCF_ATTACH_LIST.
MCF_EMAIL_FROM	Indicates the email account that sent the email, such as jsmith@company.com.

Field	***Description***
MCF_DTTM_RECV	Indicates the time that the mail server received the email. ***Warning!*** If you are using a Post Office Protocol 3 (POP3) mail server, this field may not be populated.
MCF_OFFSET_RECV	Indicates the time zone of the mail server that received this email. **Note.** This field contains the time zone of the integration gateway used to fetch this email.
MCF_DTTM_SENT	Indicates the time that the email was sent.
MCF_OFFSET_SENT	Indicates the time zone of the mail server that sent this email. **Note.** Currently, this field contains the time zone of the integration gateway used to fetch this email.
MCF_DTTM_SAVED	Records the date and time that the PeopleSoft system saved an email to the PeopleSoft database. If you use a POP3 mail server, PeopleSoft recommends using this value as the base, or starting point, for any time-sensitive transactions.
MCF_EMAIL_SENDER	This is the reply to address. This supports a mail feature in which users can send an email from one account, but when the receiver chooses to reply, the system addresses the email to a different account. For instance, sales representatives may send an email from their personal email account, as in tom_sawyer@company.com, but when customers reply, the email is addressed to sales@company.com.

Field	*Description*
MCF_EMAIL_TEXT	Represents the text that was delivered in the body of the email. Before the email is downloaded, this field contains the body of the email. After the email has been downloaded, this field is blank. Check the MCF_IS_ATT_URL value to determine whether your email text resides in this field or the attachment directory. PeopleSoft relies on the originating email client embedding encoding information to determine character set. For example, Content-type: text/plain; charset=Big5
MCF_NOTIFY_CC	Indicates the parties who were copied (carbon copied) on the email. Multiple addresses are separated by a semicolon (;).
MCF_NOTIFY_TO	Indicates the parties who received the email. Multiple addresses are separated by a semicolon (;).
MCF_UID_LIST	This contains a unique identifier created by POP3 or an IMAP4 mail server. It is guaranteed to be unique within a particular mail box. **Note.** The DeleteEmail method also uses this value when deleting a list of emails. When deleting emails, this field can contain multiple values separated by a space.

MCF_WL_SUBJECT	Contains the subject of the email. A maximum of 254 characters is allowed.
	PeopleSoft relies on the originating email client embedding encoding information to determine character set.
	For example:
	`Email Subject:  Subject: =⇒`
	`?Big5?B?uvuqwLBPqsy67qZYs/i+yS+kp⇒`
	`LDqqXik6Ldztdg=?= =?Big5?B?uvSlSKTOp⇒`
	`Eilwbr0MzCk6bVvqu2q+KTloUGkaqRP?==⇒`
	`?Big5?B?scCxUqa/v0GlwQ==?=`
	where ?Big5? is the prefixed encoding information.

MCF_RET_STATUS	Represents where the system stores the return status of a particular mail class method.
MCF_EMAIL_STATUS	This field is not currently used.

Field	*Description*
MCF_NUMROWS	Indicates the number of rows (level one child rowsets) that contain parts of a particular email. For example, if an email has two child rowsets associated with it at level one, this value would be 2. **Note.** For row one of the level zero rowset, this field reflects the number of emails in this rowset. If you fetch 10 emails, the rowset contains 11 rows, but the MCF_NUMROWS value in row 1 equals 10. Row 1 of the level zero rowset never contains an email.
MCF_EMAIL_MSG_ID	This is the globally unique identifier for this email. This ID is generated by the mail server that sent this email.
MCF_REPY_TO	This is the reply to field.
MCF_REF_IDS	Use this field for the reference header fields of the message.
MCF_REPLY_IDS	This value contains the email address that the email can reply to.
MCF_MSG_SIZE	Represents the total size of the entire email. This value includes the sizes of attachments.
MCF_EMAIL_LANG_CD	This value reflects the language code of the Integration Broker node that received this email. **Note.** PeopleSoft recommends setting up email accounts that are dedicated to one language such as, support_french@company.com and support_english@company.com. This enables you to anticipate the language of the email you read into your system. In turn, configure separate Integration Broker nodes to handle each of the languages you support. There are no language recognition procedures within the PeopleSoft system.
MCF_USER	Represents the user account (mail box) on the mail server that received the email.
MCF_SERVER	Represents the mail server that received the email.
MCF_ERROR_COUNT	Used to return the number of messages that caused errors during processing on the target connector.

Field	*Description*
MCF_QUARANTINE_COUNT	Used to return the number of messages that caused errors during processing on the target connector and were successfully moved to the quarantine folder (This value is always 0 for POP3).
MCF_CONTENT_TYPE	This is the content type of the email itself. For example, a content type could be text/plain, text/html, or multipart/alternative.
MCF_EMAIL_HEADERS	This field is used to return the email message headers. It contains the raw headers in RFC 822 format. The MCFHeaders class can be used to parse the contents of this field.

The following table describes the fields within the rowset at level one. This rowset contains the parts of the email.

Field	*Description*
MCF_ATT_URL	This is the URL of an associated attachment in the attachment directory. This URL is a relative value when this rowset is first received from the mail server and stored in the database. It only becomes a fully qualified URL once you retrieve the rowset using the MCFMailStore class. Keeping the URL a relative value enables you to make changes within the system without breaking the URL.
MCF_IS_ATT_URL	This is a Boolean value indicating whether the email part is located in the MCF_EMAIL_TEXT or on the attachment directory. If this value is 1 (true), the email part can be accessed using the MCF_ATT_URL value. If this value is 0 (false), then the email part is located in MCF_EMAIL_TEXT.
MCF_FILENAME	Represents the file name for an attachment, such as revenue.xls or resume.doc. The file name can be used as a read-only label on the page used by end users to view attachments. For example, the file name may appear next to the link that opens the attachment. **Note.** Not all parts have file names.
MCF_FILE_DATA	This field is not currently used.

Field	***Description***
MCF_EMAIL_HEADERS	This field is used to return the email part headers. It contains the raw headers in RFC 822 format. The MCFHeaders class can be used to parse the contents of this field.
MCF_EMAIL_TEXT	Contains the text content of the email part.

Note. Although the same rowset structure is used for retrieving email from the mail server and for retrieving email from the PeopleSoft database, not all of the fields apply to each situation. For example, when you first retrieve email from the mail server, the rowset does not contain the fully qualified URL to any attachments.

Note. After an email is saved to the database, always use the methods in the MailStore application package class to access the email. Do not read directly from the mail tables.

Error Messages Returned by MCFGetMail Class Methods

The GetMail Integration Broker connector returns message codes denoting status. These return codes apply to all the methods within the MCFGetMail class. The following are returned in MCF_RET_STATUS field, and by the Status property.

0

Success.

1

Connector size overflow.

The size of the email requested exceeds the value in MCF_EmSz_Conn.

The system returns the header information for this email with the return status set to 1.

See *PeopleTools 8.51 PeopleBook: PeopleSoft MultiChannel Framework*, "Configuring the Email Channel," Configuring GETMAILTARGET Properties.

2

Integration Broker threshold size overflow.

The size of the email requested exceeds the value in MCF_EmSz_IB.

This happens when the size of the XML message constructed to transfer emails from the mail server to the database exceeds the threshold value.

If there are any more emails to be processed in the current batch, the system does not fetch them from the mail server. For example, suppose your program fetches 20 emails at a time from the mail server, and this error occurred on the tenth email. In this case, the system processes the tenth email, but emails 11–20 are still be on the mail server.

See *PeopleTools 8.51 PeopleBook: PeopleSoft MultiChannel Framework*, "Configuring the Email Channel," Configuring GETMAILTARGET Properties.

3	Cannot delete all the messages from the mail server.
4	Connecting to the mail server failed.
5	The attachments to be deleted were not found.
6	At least one attachment cannot be deleted.
7	Unsupported encoding.
8	Cannot write to the attachment directory.
9	Messages were downloaded to directory and deleted from mail server. This status code occurs when an exception occurs in Integration Broker while storing the emails to the database. The exception might occur due to invalid characters in the email. If the email is left on the mail server, the same exception will occur repeatedly, preventing you from fetching further emails. Because the header might contain these invalid characters also, only MCF_RET_STATUS, MCF_MSG_SIZE, MCF_UID_LIST, date time fields, MCF_IS_ATT_URL, MCF_ATT_URL are set. There is no way to determine which email caused the exception, so the system writes all the emails to the directory. All the emails in the batch are deleted from the mail server.
10	Email content error. One example of this is unsupported encoding.
11	Returned if attempting to create a mail server folder while using the POP3 protocol.
12	Folder creation on mail server failed.
13	GetMail connector internal error.

See Chapter 23, "Mail Classes," Status, page 1156.

Mail Classes Constructors

The following section provides examples of the mail class constructors.

MCFBodyPart

Example

```
Local PT_MCF_MAIL:MCFBodyPart &text = create PT_MCF_MAIL:MCFBodyPart();
```


MCFGetMail

Example

```
Local MCFGetMail &gm = create MCFGetMail();
```

MCFOutboundEmail

Example

```
Local PT_MCF_MAIL:MCFOutboundEmail &eMail =
create PT_MCF_MAIL:MCFOutboundEmail();
```

MCFMailStore

Example

```
Local MCFGetMail &sm = create MCFMailStore();
```

SMTPSession

Example

```
Local PT_MCF_MAIL:SMTPSession &commonSession =
create PT_MCF_MAIL:SMTPSession();
```

MCFBodyPart Class

Use the MCFBodyPart class to send email with attachments, HTML, or other non-standard text information.

MCFBodyPart Class Methods

In the following, we discuss the MCFBodyPart class methods. The methods are described in alphabetical order.

AddHeader

Syntax

AddHeader(*HeaderName*,*HeaderValue*)

Description

Use the AddHeader method to add a header to the body part. This method allows for email server customizations. Some commonly used headers are already exposed as properties, such as ContentType and ContentLanguage. Advanced applications can adapt this technique to meet their own requirements. These headers can be either standard SMTP headers or custom headers starting with "X-".

Parameters

Parameter	*Description*
HeaderName	Specify the name of the header that you want to add. This parameter takes a string value.
HeaderValue	Specify the actual text of the header, as a string.

Returns

None.

Example

```
   Local PT_MCF_MAIL:MCFOutboundEmail &email = create PT_MCF_MAIL:MCFOutboundEmail⇒
();
   Local string &TestName = "Custom Header";

   &email.From = &def.From;
   &email.Recipients = &def.Recipients;
   &email.SMTPServer = &def.SMTPServer;
   &email.Subject = &TestName;
   &email.Text = &TestName;

   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP2");
   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP3");
   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP4");
   &res = &email.Send();
```

See Also

Chapter 23, "Mail Classes," GetHeader, page 1118; Chapter 23, "Mail Classes," GetHeaderCount, page 1118; Chapter 23, "Mail Classes," GetHeaderName, page 1119; Chapter 23, "Mail Classes," GetHeaderNames, page 1120; Chapter 23, "Mail Classes," GetHeaderValues, page 1120 and Chapter 23, "Mail Classes," GetUnparsedHeaders, page 1121

GetHeader

Syntax

`GetHeader``(`*`HeaderName`*`)`

Description

Use the GetHeader method to return the value of the specified header.

This method is primarily used with inbound email messages.

Parameters

Parameter	*Description*
HeaderName	Specify the name of the header that you want to access the values for, as a string. The matching of header names is case insensitive.

Returns

An array of string containing the header values. If the header is not present, returns an array of length zero.

See Also

Chapter 8, "Array Class," page 257

Chapter 23, "Mail Classes," AddHeader, page 1117; Chapter 23, "Mail Classes," GetHeaderCount, page 1118; Chapter 23, "Mail Classes," GetHeaderName, page 1119; Chapter 23, "Mail Classes," GetHeaderNames, page 1120; Chapter 23, "Mail Classes," GetHeaderValues, page 1120 and Chapter 23, "Mail Classes," GetUnparsedHeaders, page 1121

GetHeaderCount

Syntax

`GetHeaderCount``()`

Description

Use the GetHeaderCount method to return the number of headers.

Parameters

None.

Returns

An integer, representing the number of headers present.

See Also

Chapter 23, "Mail Classes," AddHeader, page 1117; Chapter 23, "Mail Classes," GetHeader, page 1118; Chapter 23, "Mail Classes," GetHeaderName, page 1119; Chapter 23, "Mail Classes," GetHeaderNames, page 1120; Chapter 23, "Mail Classes," GetHeaderValues, page 1120 and Chapter 23, "Mail Classes," GetUnparsedHeaders, page 1121

GetHeaderName

Syntax

`GetHeaderName`*`(Index)`*

Description

Use the GetHeaderName to return the name of the header that is located at *Index.*

Parameters

Parameter	***Description***
Index	Specify the numeric position of the header you want to access.

Returns

A string.

See Also

Chapter 23, "Mail Classes," AddHeader, page 1117; Chapter 23, "Mail Classes," GetHeader, page 1118; Chapter 23, "Mail Classes," GetHeaderCount, page 1118; Chapter 23, "Mail Classes," GetHeaderNames, page 1120; Chapter 23, "Mail Classes," GetHeaderValues, page 1120 and Chapter 23, "Mail Classes," GetUnparsedHeaders, page 1121

GetHeaderNames

Syntax

`GetHeaderNames``()`

Description

Use the GetHeaderNames method to return an array containing the names of all the headers.

This method is primarily used with inbound email messages.

Parameters

None.

Returns

An array of string.

See Also

Chapter 23, "Mail Classes," AddHeader, page 1117; Chapter 23, "Mail Classes," GetHeader, page 1118; Chapter 23, "Mail Classes," GetHeaderCount, page 1118; Chapter 23, "Mail Classes," GetHeaderName, page 1119; Chapter 23, "Mail Classes," GetHeaderValues, page 1120 and Chapter 23, "Mail Classes," GetUnparsedHeaders, page 1121

GetHeaderValues

Syntax

`GetHeaderValues``(`*`Index`*`)`

Description

Use the GetHeaderValues method to return the value for the header located at the position specified by *Index.*

Parameters

Parameter	*Description*
Index	Specify the numeric position of the header whose value you want to access.

Returns

An array of string.

See Also

Chapter 23, "Mail Classes," AddHeader, page 1117; Chapter 23, "Mail Classes," GetHeader, page 1118; Chapter 23, "Mail Classes," GetHeaderCount, page 1118; Chapter 23, "Mail Classes," GetHeaderName, page 1119; Chapter 23, "Mail Classes," GetHeaderNames, page 1120 and Chapter 23, "Mail Classes," GetUnparsedHeaders, page 1121

GetUnparsedHeaders

Syntax

GetUnparsedHeaders()

Description

Use the GetUnparsedHeaders method to return all of the headers in the body part without any formatting occurring.

Parameters

None.

Returns

A string containing the headers.

See Also

Chapter 23, "Mail Classes," AddHeader, page 1117; Chapter 23, "Mail Classes," GetHeader, page 1118; Chapter 23, "Mail Classes," GetHeaderCount, page 1118; Chapter 23, "Mail Classes," GetHeaderName, page 1119; Chapter 23, "Mail Classes," GetHeaderNames, page 1120 and Chapter 23, "Mail Classes," GetHeaderValues, page 1120

SetAttachmentContent

Syntax

```
SetAttachmentContent({FilePath | FileURL}, FilePathType,FileName,FileDescr,
OverrideContentType,OverrideCharset)
```

Description

Use the SetAttachmentContent method to specify the body (content) of this body part from a file.

Use the *FilePathType* parameter to specify if the file path is relative or absolute. If it's an absolute path, the file path could be a file on the local machine, a network folder, or a fully-qualified URL.

Parameters

Parameter	***Description***
FilePath | *FileURL*	You can either specify the file path to the file, or a URL to the file. Depending on the *FilePathType* parameter, you may specify either a relative or absolute file path to the file. This parameter should include the file name and extension of the file. If you specify a URL address where the file is located, it must be an absolute URL. This URL should include the actual name of the file. If you specify a URL address, you must specify the *FilePathType* parameter as %FilePath_Absolute.
FilePathType	Specify the path type for the file, whether it is an absolute or relative path. The values are: • %FilePath_Absolute — the file path is an absolute path. • %FilePath_Relative — the file path is a relative path.
FileName	Specify the name of the file that you want to attach, as a string. You must include the file extension as well.
Description	Specify a description of the file.
OverrideContentType	The system detects the content type of the attachment using the file location and name. Specifying *OverrideContentType* overrides the system detected content type. The character set can also be included in the ContentType. For example, "text/plain; charset=US-ASCII"
OverrideCharset	Specify a character set to be used to override the existing character set.

Returns

None.

Example

The following uses an absolute path to set the content:

```
&image.SetAttachmentContent("///file:C:/User/Documentum/XML%20Applications/proddoc⇒
/peoplebook_upc/peoplebook_upc.dtd",
 %FilePath_Absolute, "sample.jpg", "This is a sample image!", "", "");
```

The following uses a relative path to set the content:

```
&attach1.SetAttachmentContent("Ocean Wave.jpg", %FilePath_Relative,
"Ocean Wave.jpg", "Ocean Wave", "", "");
```

The following uses a URL to set the content:

```
&attach1.SetAttachmentContent("http://www.yahoo.com/members_agreement",
 %FilePath_Absolute, "hotmail.htm", "Hotmail Home page", "", "");
```

See Also

Chapter 23, "Mail Classes," AddAttachment, page 1178

MCFBodyPart Class Properties

In this section we discuss the MCFBodyPart class properties. The properties are described in alphabetical order.

AttachmentURL

Description

Use this property to specify the URL for the attachment for this body part, as a string.

The URL must be an absolute URL.

This property is read-write.

Charset

Description

Use this property to specify the character set for the text or the attachment.

You can also specify this property using the SetAttachmentContent method, or the ContentType property.

This property is read-write.

See Also

Chapter 23, "Mail Classes," ContentType, page 1124

Chapter 23, "Mail Classes," SetAttachmentContent, page 1122

ContentType

Description

Use this property to specify the content type of this body part.

You can specify the content type with the SetAttachmentContent method.

You can also use this property to specify the character set. For example, "text/plain; charset=US-ASCII".

This property is read-write.

See Also

Chapter 23, "Mail Classes," SetAttachmentContent, page 1122

Chapter 23, "Mail Classes," Charset, page 1124

Description

Description

Use this property to specify a description of the attachment. You should only use this property if the content is an attachment.

This property is read-write.

Disposition

Description

Use this property to specify how the body part is presented in the received mail.

Some email clients ignore the setting of this property and present body parts either inline or as file attachments.

Valid values for this property are:

Value	*Description*
Attachment	The body part is shown as a file attachment.
Inline	The body part is shown in the body itself.

This property is read-write.

Filename

Description

If you are adding an attachment to the email, use this property to specify the name of the file.

PeopleSoft recommends that you keep the file extension specified with this property the same as the original extension found in the file path, otherwise client applications may not be able to display it properly.

This property is read-write.

FilePath

Description

Specify the path for the file that contains the contents of this email.

Whether this is a relative or absolute path depends on the FilePathType property.

You can also specify a URL to the file using this property, if the FilePathType property is specified as %FilePath_Absolute.

If you specify a value for this property, the 'Text' content is ignored.

This property is read-write.

See Also

Chapter 23, "Mail Classes," FilePathType, page 1126

FilePathType

Description

Use this property to specify whether the path specified with the FilePath property is a relative or absolute path. The values for this property are:

Value	***Description***
%FilePath_Relative	The file path specified with the FilePath property is a relative path.
%FilePath_Absolute	The file path specified with the FilePath property is either an absolute path to a file, or a URL to a file.

If you specify a relative path, the file must be available in the FILES folder of application server folder.

If you specify an absolute path, the file could be on the local machine, in any network folder, or a URL.

This property is read-write.

IsAttachment

Description

This property indicates if the body part is an attachment or not. This property returns a Boolean value: true if it is an attachment, false otherwise.

This property is read-only.

MultiPart

Description

A bodypart can contain simple text, one attachment, or a MultiPart object.

If you have assigned a MultiPart object using this property, the text and attachment related properties are ignored.

This property is read-write.

Example

```
Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
&mp.SubType = "alternative; differences=Content-type";

&mp.AddBodyPart(&text);
&mp.AddBodyPart(&html);

&email.MultiPart = &mp;
```

Text

Description

Use this property to specify the text for the email.

Use the SetAttachmentContent method to specify image or other types of attachments. For a more complex bodypart, build a multipart and set the Multipart property of MCFBodyPart.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SetAttachmentContent, page 1122

MCFEmail Class

This class inherits many properties from the MCFPart class, and is the superclass class for both the MCFOutboundEmail and MCFInboundEmail classes. Generally, you only use the MCFOutboundEmail and MCFInboundEmail classes. You should only use the MCFEmail class if you are extending the PT_MCF_MAIL application package.

MCFEmail Class Properties

In this section, we discuss the MCFEmail class properties. The properties are discussed in alphabetical order.

BCC

Description

Use this property to specify comma-separated list of addresses of all the blind carbon copy recipients of this email. (A blind carbon copy is a copy of the email that contains all of the text of the email, but does not show any of the other email addresses to which this email was sent.)

This property is read-write.

See Also

Chapter 23, "Mail Classes," CC, page 1128

BounceTo

Description

Use this property to specify the email address the system should direct all bounced mail to.

This property is read-write.

CC

Description

Use this property to specify a comma-separated list of addresses that copies of this email are to be sent to (carbon copy.)

This property is read-write.

See Also

Chapter 23, "Mail Classes," BCC, page 1128

From

Description

Use this property to specify the email address of the person sending the email.

You can specify more than one address as from in a comma-separated list.

This property is read-write.

See Also

Chapter 23, "Mail Classes," Recipients, page 1130; Chapter 23, "Mail Classes," ReplyTo, page 1130 and Chapter 23, "Mail Classes," Sender, page 1131

Importance

Description

Use this property to specify the value of the importance header field.

The importance can be set to the following:

- low
- normal
- high

If the Priority property is not set, the system automatically sets it to the corresponding values like 5, 3 and 1.

This property is read-write.

Priority

Description

Use this property to specify the priority of this email.

The values are:

1. Highest Priority
2. High Priority
3. Normal
4. Low Priority
5. Lowest Priority

The default value is 3, that is, normal priority.

This value set with this property is used to set to a header X-Priority field, as this is the most common but non-standard field to show priority. In addition, the corresponding values are set in two header fields: X-MSMail-Priority and Priority. The headers X-Priority or X-MSMail-Priority are set to the corresponding value only if the user does not specify a value.

This property is read-write.

Recipients

Description

Use this property to specify the email addresses of all the main recipients to whom the email is being sent. All the addresses are specified in a comma-separated string.

This property is read-write.

RefIDs

Description

Use this property to specify a value for the REFERENCES header field of the message.

This property is read-write.

ReplyIDs

Description

Use this property to specify a value for the IN-REPLY-TO header field of the message.

This property is read-write.

ReplyTo

Description

Use this property to specify the email address where the reply should be sent. You do not need to specify a value for this property if the value is the same as the From property.

This property is read-write.

See Also

Chapter 23, "Mail Classes," From, page 1128

Sender

Description

Use this property to specify the address of the author of the message. You do not need to specify a value for this property if the value for the From property is the same.

This property is read-write.

See Also

Chapter 23, "Mail Classes," From, page 1128

Sensitivity

Description

Use this property to specify the value of the sensitivity header field.

The values are:

- personal
- private
- company-confidential

This property is read-write.

Subject

Description

Use this property to specify the subject of the email. A subject can only contain 254 characters.

This property is read-write.

Text

Description

Use this property to specify the text of the email.

This property is read-write.

MCFGetMail Class Methods

This section discusses MCFGetMail class methods.

CreateQuarantineFolder

Syntax

`CreateQuarantineFolder``()`

Description

Use the CreateQuarantineFolder method to create a quarantine folder on the server. Specify the name of the folder using the QuarantineFolder property

Parameters

None.

Returns

A Boolean value; true if the folder is created, or if it already exists on the server, false otherwise.

See Also

Chapter 23, "Mail Classes," QuarantineFolder, page 1145 and Chapter 23, "Mail Classes," QuarantineCount, page 1145

GetCount

Syntax

`GetCount``()`

Description

Use the GetCount method to determine the total number of emails currently stored in a particular email account.

This method does not distinguish between read and unread emails.

This method uses the connection properties set by SetMCFEmail.

If you want to determine the number of emails for an account, and to set your own connection properties, use the GetEmailCount method instead.

Parameters

None.

Returns

Returns the total number of emails stored on the mail server for a particular email account. If the method returns –4, the system could not connect to the mail server using the user name and password passed in the parameter list.

See Also

Chapter 23, "Mail Classes," GetEmailCount, page 1133 and Chapter 23, "Mail Classes," SetMCFEmail, page 1142

GetEmailCount

Syntax

`GetEmailCount``(`*`user,password,server, node`*`)`

Description

Use GetEmailCount to determine the number of emails currently stored in a particular email account. GetEmailCount does not distinguish between read and unread email. You can use this value to set limits on loops in your program.

If you want to determine the number of emails for an account using the connection properties set with SetMCFEmail, use the GetCount method instead.

Parameters

Parameter	*Description*
user	The user name on the mail server, such as "support" or "john_doe".
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.

Parameter	*Description*
node	The Integration Broker node on which the request runs.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, Integration Broker returns an error.

Returns

Returns the total number of emails stored on the mail server for a particular email account. If the method returns –4, the system could not connect to the mail server using the user name and password passed in the parameter list.

Example

```
Local number &email_count;

&email_count = &gm.GetEmailCount(&username, &passwd, &mailserver,
&node);
```

See Also

Chapter 23, "Mail Classes," GetCount, page 1132 and Chapter 23, "Mail Classes," SetMCFEmail, page 1142

ReadAllEmailHeadersWithAttach

Syntax

ReadAllEmailHeadersWithAttach(*user*, *password*, *server*, *node*)

Description

Use ReadAllEmailHeadersWithAttach to read the headers of all of the emails and a list of associated attachments that are currently stored on the mail server for a particular email account. The details about the emails are returned in the level zero rowset. The actual emails and any associated attachments *are not* returned.

Use this method as a screening method to determine whether to store an email in your PeopleSoft system. For example, if you determine that an email is junk mail, you could delete it from your mail server. After you determine the allowable email through header information, you can retrieve the physical email and attachments of the allowable emails.

Parameters

Parameter	*Description*
user	The user name on the mail server, such as "support" or "john_doe".
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The Integration Broker node on which the request runs.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, Integration Broker returns an error.

Returns

This method returns rowsets containing email headers.

Example

```
import PT_MCF_MAIL:*;

Local MCFGetMail &gm;

Local Rowset &emails_rs;

&emails_rs = &gm.ReadAllEmailHeadersWithAttach(&username, &passwd, &mailserver
&node);
```

ReadEmails

Syntax

ReadEmails*(Count)*

Description

Use the ReadEmails method to read the specified number of emails for this mailbox.

This method uses the connection properties set by SetMCFEmail.

The overall status of the success of this method is available in the Status property.

Parameters

Parameter	*Description*
Count	Specify the number of emails you want to read.

Returns

An array of MCFInboundEmail.

See Also

Chapter 23, "Mail Classes," MCFInboundEmail Class Methods, page 1147

Chapter 23, "Mail Classes," Status, page 1156

ReadEmailsWithAttach

Syntax

`ReadEmailsWithAttach``(User,password,server,node,count)`

Description

The ReadEmailsWithAttach method reads the number of emails specified by *count* for a particular user account. If attachments are associated with an email, this method reads the attachments as well.

Parameters

Parameter	*Description*
User	Specify the user account that contains the emails you want to access, such as "support" or "john_doe".
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The Integration Broker node on which the request runs.
count	Specifies the number of emails to read. If count = 0, the method reads *all* the emails of the user account. If count > 0, the method reads the number specified. For example, if count = 10, the method reads 10 emails.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, Integration Broker returns an error.

Returns

This method returns an email within a rowset. level one of the rowset contains the email parts. level zero row 1 only contains values for MCF_NUMROWS and MCF_RET_STATUS.

Example

```
import PT_MCF_MAIL:*;

Local MCFGetMail &gm;

Local Rowset &emails_rs;

&emails_rs = &gm.ReadEmailsWithAttach(&username, &passwd, &mailserver
&node, &email_count);
```

See Also

Chapter 23, "Mail Classes," ReadEmailWithAttach, page 1138

ReadEmailsWithUID

Syntax

`ReadEmailsWithUID`*`(UID)`*

Description

Use the ReadEmailsWithUID method to read and return an email based on a unique identifier *UID*. This method uses the connection properties set by SetMCFEmail. Emails that have an attachment are also read.

Parameters

Parameter	*Description*
UID_List	The UID is the unique identifier of the email. This parameter is only unique within a particular user account. POP3 and IMAP 4 mail servers automatically create a unique identifier for an email when the server receives the email. The list can contain multiple values, each separated by a space.

Returns

A reference to an array of MCFInboundEmail.

See Also

Chapter 23, "Mail Classes," MCFInboundEmail Class Methods, page 1147

ReadEmailWithAttach

Syntax

`ReadEmailWithAttach``(`*`User,password,server, node,UID`*`)`

Description

Use the ReadEmailWithAttach method to read and return an email based on a unique identifier *(UID)* for the specific email to be read. If attachments are associated with this email, ReadEmailWithAttach fetches those as well.

Parameters

Parameter	***Description***
user	The user name on the mail server, such as "support" or "john_doe".
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The Integration Broker node on which the request runs.
UID	The unique identifier of the email. This parameter is only unique within a particular user account. POP3 and IMAP 4 mail servers automatically create a unique identifier for an email when the server receives the email. You may specify more than one UID, each separated by a space.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, Integration Broker returns an error.

Returns

This method returns an email within a rowset. level one of the rowset contains the email parts. level zero row 1 only contains values for MCF_NUMROWS and MCF_RET_STATUS.

Example

```
import PT_MCF_MAIL:*;

Local MCFGetMail &gm;

Local Rowset &emails_rs;

&emails_rs = &gm.ReadEmailWithAttach(&username, &passwd, &mailserver, &node, &UID);
```

See Also

Chapter 23, "Mail Classes," ReadEmailsWithAttach, page 1136

ReadHeaders

Syntax

ReadHeaders()

Description

Use the ReadHeaders method to read the headers of all emails currently on the server. This method also returns the attachment information for all emails (names and sizes.)

This method uses the connection properties set by SetMCFEmail.

The overall success of the method is available using the Status property.

Parameters

None.

Returns

An array of MCFInboundEmail objects.

See Also

Chapter 23, "Mail Classes," MCFInboundEmail Class Methods, page 1147

Chapter 23, "Mail Classes," SetMCFEmail, page 1142

RemoveEmail

Syntax

```
RemoveEmail(user,password,server, node,UID list)
```

Description

Based on values in the unique identifier list, RemoveEmail deletes emails from the user account on the mail server. It is important to delete emails from the mail server once they are read into PeopleSoft. If you do not delete emails from the mail server after they are read into PeopleSoft, the rowset contains the same collection of email the next time you retrieve email from the mail server.

Parameters

Parameter	***Description***
user	The user name on the mail server, such as "support" or "john_doe".
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The Integration Broker node on which the request runs.
UID list	The UID list contains the unique identifier values for the emails that need to be deleted from the mail server. The list can contain multiple values separated by a space.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, the Integration Broker returns an error.

Returns

A Boolean value, indicating the status of the deletion process. If the email has been successfully, this method returns True. Otherwise the method returns False.

Example

```
Local MCFGetMail &GM;
Local boolean &del_status;
Local string &uid_list = "";
Local string &msg_str;

   If (&uid_list <> "") Then
      &del_status = &GM.RemoveEmail(&user, &password, &server, &ibnode, &uid_list);
      &msg_str = "Email(s) deleted from the Mail Server with UID = " | &uid_list;
      MessageBox(0, "", 0, 0, &msg_str);
   End-If;
```

See Also

Chapter 23, "Mail Classes," RemoveEmails, page 1141

RemoveEmails

Syntax

RemoveEmails*(UID_List)*

Description

Use the RemoveEmails method to delete emails from an account on the mail server, based on the *UID_list.* It is important to delete emails from the mail server once they are read into PeopleSoft. If you do not delete emails from the mail server after they are read into PeopleSoft, the rowset contains the same collection of email the next time you retrieve email from the mail server.

Parameters

Parameter	*Description*
UID_List	The UID list contains the unique identifier values for the emails that need to be deleted from the mail server. The list can contain multiple values separated by a space.

Returns

An array of string containing UIDs of the messages that were successfully removed, separated by semicolons.

See Also

Chapter 23, "Mail Classes," RemoveEmail, page 1140

SetMCFEmail

Syntax

`SetMCFEmail`*`(user,password,server, node)`*

Description

Use the SetMCFEmail method to set the connection properties for subsequent method calls.

This method is a shortcut for setting the UserId, Password, MailServer, and IBNode properties.

Parameters

Parameter	***Description***
user	The user name on the mail server, such as "support" or "john_doe".
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The Integration Broker node on which the request runs.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, the integration broker returns an error.

Returns

None.

See Also

Chapter 23, "Mail Classes," GetCount, page 1132; Chapter 23, "Mail Classes," ReadEmails, page 1135; Chapter 23, "Mail Classes," ReadEmailsWithUID, page 1137 and Chapter 23, "Mail Classes," ReadHeaders, page 1139

MCFGetMail Class Properties

In this section we discuss the MCFGetMail class properties. The properties are described in alphabetical order.

AttachmentRoot

Description

Use this property to specify the root directory where the attachments are stored. If not specified, the value from the node is used.

Valid metastrings that can be included in the value for this property are:

%CURRDATE%	Current date in YYYYMMDD format
%CURRHOUR%	Current hour in 24–hour format (00-23)
%DBNAME%	Current value of %DbName system variable.
%OPRID%	Current value of %UserId system variable.

This property is read-write.

Example

To have the attachment root directory change hourly, you could use the following code example:

```
&MyEmail.AttachmentRoot = "c:\temp\%CURRDATE%\%CURRHOUR%\";
```

ContentTypes

Description

Use this property to specify content types that should be returned in the message or part text rather than stored in the directory that contains the attachments. Specify content types in a comma separated list. For example: `"text/html,text/xml"`.

Note. `"Text/plain"` is always returned in the text field and does not need to be included.

If not specified, the value from the node is used.

This property is read-write.

ErrorCount

Description

This property returns the number of messages that caused errors during processing on the target connector.

You should use this property after you use the ReadHeaders, ReadEmails, ReadEmailsWithUID, or RemoveEmails methods.

This property is read-only.

See Also

Chapter 23, "Mail Classes," ReadHeaders, page 1139; Chapter 23, "Mail Classes," ReadEmails, page 1135; Chapter 23, "Mail Classes," ReadEmailsWithUID, page 1137 and Chapter 23, "Mail Classes," RemoveEmails, page 1141

IBNode

Description

This property returns the name of the Integration Broker node on which a request ran, such as MCF_GetMail.

This property is set by the SetMCFEmail method. It is used by ReadHeaders, ReadEmails, ReadEmailsWithUID, RemoveEmails, and CreateQuarantineFolder methods.

This property is read-write.

See Also

Chapter 23, "Mail Classes," ReadHeaders, page 1139; Chapter 23, "Mail Classes," ReadEmails, page 1135; Chapter 23, "Mail Classes," ReadEmailsWithUID, page 1137; Chapter 23, "Mail Classes," RemoveEmails, page 1141 and Chapter 23, "Mail Classes," CreateQuarantineFolder, page 1132

MailServer

Description

This property returns the name of the mail server that received the email.

This property is set by the SetMCFEmail method. It is used by ReadHeaders, ReadEmails, ReadEmailsWithUID, RemoveEmails, and CreateQuarantineFolder methods.

This property is read-write.

See Also

Chapter 23, "Mail Classes," ReadHeaders, page 1139; Chapter 23, "Mail Classes," ReadEmails, page 1135; Chapter 23, "Mail Classes," ReadEmailsWithUID, page 1137; Chapter 23, "Mail Classes," RemoveEmails, page 1141 and Chapter 23, "Mail Classes," CreateQuarantineFolder, page 1132

Password

Description

This property returns the password associated with the userID that was used to get the email.

This property is set by the SetMCFEmail method. It is used by ReadHeaders, ReadEmails, ReadEmailsWithUID, RemoveEmails, and CreateQuarantineFolder methods.

This property is read-write.

See Also

Chapter 23, "Mail Classes," ReadHeaders, page 1139; Chapter 23, "Mail Classes," ReadEmails, page 1135; Chapter 23, "Mail Classes," ReadEmailsWithUID, page 1137; Chapter 23, "Mail Classes," RemoveEmails, page 1141 and Chapter 23, "Mail Classes," CreateQuarantineFolder, page 1132

QuarantineCount

Description

This property returns the number of email messages that caused errors during processing on the target connector and were successfully moved to the quarantine folder.

This property is valid after calls to ReadHeaders, ReadEmails, ReadEmailsWithUID, and RemoveEmails.

This property is read-only.

See Also

Chapter 23, "Mail Classes," QuarantineFolder, page 1145

Chapter 23, "Mail Classes," ReadHeaders, page 1139; Chapter 23, "Mail Classes," ReadEmails, page 1135; Chapter 23, "Mail Classes," ReadEmailsWithUID, page 1137 and Chapter 23, "Mail Classes," RemoveEmails, page 1141

QuarantineFolder

Description

Use this property to specify the name of the quarantine folder. If not specified, the value from the node will be used.

The name must adhere to whatever naming restrictions are imposed by the mail server in use. There is no built-in validation of the folder name.

The following types of errors can cause an email to be moved to the quarantine folder:

- Connector size overflow.
- Unsupported encoding.
- Unknown Java exception.
- Javamail content error.
- No attachment directory.
- Mail parse exception.

This property is read-write.

See Also

Chapter 23, "Mail Classes," QuarantineCount, page 1145

Status

Description

This property returns the status of the last mail server operation.

See Chapter 23, "Mail Classes," Error Messages Returned by MCFGetMail Class Methods, page 1114.

This property is valid after calls to ReadHeaders, ReadEmails, ReadEmailsWithUID, and RemoveEmails.

This property is read-only.

UserID

Description

This property returns the user ID (mail box) on the mail server that received the email.

This property is set by SetMCFEmail. It is used by ReadHeaders, ReadEmails, ReadEmailsWithUID,RemoveEmails, and CreateQuarantineFolder methods.

This property is read-write.

MCFInboundEmail Class

This class inherits many properties from the MCFEmail class, which in turn inherits many properties from the MCFPart class. Generally, you use only the subclasses, MCFInboundEmail and MCFOutboundEmail, and not the superclasses.

MCFInboundEmail Class Methods

In this section, we discuss the MCFInboundEmail class methods. The methods are described in alphabetical order.

DumpToFile

Syntax

`DumpToFile`*`(&File)`*

Description

Use the DumpToFile method to write information from the email to an already instantiated file.

This method copies the information from the following email properties, placing each on a separate line:

- ContentType
- IsAttachment
- AttachmentURL
- Text

Whether the information is appended to the bottom of the file or if it overwrites the file depends on how you instantiated the file object.

Parameters

Parameter	***Description***
&File	Specify an already instantiated file object that you want to write the email data to.

Returns

None.

Example

```
Local File &BI_FILE;
&BI_FILE = GetFile("mcfdata.out", "w", "a");

Local PT_MCF_MAIL:MCFGetMail &gm;
Local number &ret_status, &nemails, &i;
&gm = create PT_MCF_MAIL:MCFGetMail();

Local array of PT_MCF_MAIL:MCFInboundEmail &emails;
&gm.SetMCFEmail(&user, &password, &server, &ibnode);

&emails = &gm.ReadEmails(0);
If (&gm.StatusCheck(&BI_FILE) = False) Then
   Return;
End-If;

&nemails = &emails.Len;
&ret_status = &gm.Status;
If &nemails > 0 Then
   For &i = 1 To &nemails
      &BI_FILE.WriteLine(MsgGetText(162, 1615, "Message Not Found", &i));
      &emails [&i].DumpToFile(&BI_FILE);
   End-For;
End-If;
```

See Also

Chapter 19, "File Class," page 975

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

GetAttachments

Syntax

GetAttachments()

Description

Use the GetAttachments method to return the attachments with this email.

Parameters

None.

Returns

An array of MCFBodyPart objects. The methods returns an array of length 0 if there are no attachments

See Also

Chapter 8, "Array Class," page 257

Chapter 23, "Mail Classes," MCFBodyPart Class Methods, page 1116

GetFrom

Syntax

GetFrom()

Description

Use the GetFrom method to split the semicolon delimited list of addresses in the From property into an array of addresses.

Parameters

None.

Returns

An array of string. Each item in the array contains an email address.

See Also

Chapter 23, "Mail Classes," From, page 1128

GetParts

Syntax

GetParts()

Description

Use the GetParts method to return all of this email's parts.

Parameters

None.

Returns

An array of MCFBodyPart objects.

See Also

Chapter 23, "Mail Classes," MCFBodyPart Class Methods, page 1116

GetSender

Syntax

`GetSender``()`

Description

Use the GetSender method to split the semicolon delimited list of addresses in the Sender property into an array of addresses.

Parameters

None.

Returns

An array of string.

See Also

Chapter 23, "Mail Classes," Sender, page 1131

Chapter 23, "Mail Classes," GetFrom, page 1149

ReadFromDatabase

Syntax

ReadFromDatabase*(Email_ID)*

Description

Use the ReadFromDatabase method to load a saved email from the database into this instance of email.

Parameters

Parameter	*Description*
Email_ID	Specify the ID of the email, as a number.

Returns

A Boolean: true if the email is restored successfully, false otherwise.

See Also

Chapter 23, "Mail Classes," SaveToDatabase, page 1151

SaveToDatabase

Syntax

SaveToDatabase()

Description

Use the SaveToDatabase method to save this instance of email to the database.

Parameters

None.

Returns

A number, representing the email_ID of the saved email.

See Also

Chapter 23, "Mail Classes," ReadFromDatabase, page 1151

MCFInboundEmail Class Properties

In this section, we discuss the MCFInboundEmail class properties. The properties are described in alphabetical order.

AttachList

Description

This property returns the list of all the file names that are attachments for a particular email.

Multiple items are separated by a semicolon (;).

This property is read-write.

See Also

Chapter 23, "Mail Classes," AttachSizes, page 1152

AttachSizes

Description

This property returns the sizes of the attachments associated with an email.

This property takes a string value, *not* a numeric value. Multiple values are separated by a semicolon (;).

The attachment sizes appear in the same order as the file names in AttachList.

This property is read-write.

See Also

Chapter 23, "Mail Classes," AttachList, page 1152

DttmReceived

Description

Use this property to return the date and time an email was received.

Warning! If you are using a Post Office Protocol 3 (POP3) mail server, this property may not be populated.

This property takes a datetime value.

This property is read-write.

See Also

Chapter 23, "Mail Classes," DttmSaved, page 1153 and Chapter 23, "Mail Classes," DttmSent, page 1153

DttmSaved

Description

This property returns the date and time that the PeopleSoft system saved an email to the database.

If you use a POP3 mail server, PeopleSoft recommends using this value as the base, or starting point, for any time-sensitive transactions.

This property takes a datetime value.

This property is read-write.

See Also

Chapter 23, "Mail Classes," DttmReceived, page 1153 and Chapter 23, "Mail Classes," DttmSent, page 1153

DttmSent

Description

This property returns the date time an email was sent.

This property takes a datetime value.

This property is read-write.

See Also

Chapter 23, "Mail Classes," DttmReceived, page 1153 and Chapter 23, "Mail Classes," DttmSent, page 1153

IBNode

Description

This property returns the name of the Integration Broker node that received the email, such as MCF_GetMail.

The system uses this value to construct the fully qualified URL required for viewing attachments.

This property is read-write.

Language

Description

This value reflects the language code of the Integration Broker node that received this email.

Note. PeopleSoft recommends setting up email accounts that are dedicated to one language such as, support_french@company.com and support_english@company.com. This enables you to anticipate the language of the email you read into your system. In turn, configure separate Integration Broker nodes to handle each of the languages you support. There are no language recognition procedures within the PeopleSoft system.

This property is read-write.

MessageID

Description

Use this property to return the globally unique identifier for this email.

This ID is generated by the mail server that sent this email.

NotifyCC

Description

This property returns a list of the parties who were copied (carbon copied) on the email. Multiple addresses are separated by a semicolon (;).

This property is read-write.

NotifyTo

Description

This property returns a list of the parties who received the email. Multiple addresses are separated by a semicolon (;).

This property is read-write.

OffsetReceived

Description

This property returns time zone of the mail server that received this email, as a number.

Note. This property contains the time zone of the integration gateway used to fetch this email.

This property is read-write.

See Also

Chapter 23, "Mail Classes," OffsetSent, page 1155

OffsetSent

Description

This property returns the time zone of the mail server that sent this email, as a string.

Note. Currently, this property contains the time zone of the integration gateway used to fetch this email.

This property is read-write.

See Also

Chapter 23, "Mail Classes," OffsetReceived, page 1155

Server

Description

This property specifies the name of the mail server that received the email.

This property is read-write.

Size

Description

This property returns the total size of the entire email. This value includes the sizes of attachments.

This property is read-write.

Status

Description

This property returns the status of an individual email.

This property is read-only.

UID

Description

This property returns the unique identifier of the email.

This property is only unique within a particular user account. POP3 and IMAP 4 mail servers automatically create a unique identifier for an email when the server receives the email.

This property is read-write.

User

Description

This property returns the user account (mail box) on the mail server that received the email.

This property is read-write.

MCFMailStore Class

Use the MCFMailStore class to write emails to the PeopleSoft database and to retrieve emails from the PeopleSoft database.

Note. When writing emails to the PeopleSoft database, it is assumed that you have previously used the MCFGetMail class to retrieve email into memory. You use the MCFMailStore class immediately after to save the rowsets to the database.

MCFMailStore Import Statements

Here is the import statement:

```
import PT_MCF_MAIL:MCFMailStore;
```

MCFMailStore Methods

This section discusses MCFMailStore class methods.

AuthorizeEmailAttach

Syntax

AuthorizeEmailAttach(*email ID,authentication name*, *authentication type, operation*)

Description

Use AuthorizeEmailAttach to authorize a user or a PeopleSoft role to view attachments associated with an email. By default, no user or role is authorized to view an email; you must explicitly grant authorization with this method. This method authorizes email and the associated attachments one at a time.

Note. If you specify ALL as the user, *any* user can view the attachments for an email.

Parameters

Parameter	***Description***
email ID	The unique email ID for the email. This is the PeopleSoft email ID, not the ID issued by the mail server.

Parameter	***Description***
authentication name	The name of the PeopleSoft user profile or role that needs to be authorized to view the attachments.
authentication type	Specify the type security definition entered as the authentication name. • Specify *2* to indicate a user profile. • Specify*3* to indicate a role.
operation	Specify the type of authorization operation. The following list contains the supported operations. You either add or delete user or role authorization. • 1 = ADD • 2 = DELETE

Returns

Returns a Boolean value: True for success, False otherwise.

Example

```
&ms = create MCFMailStore();
&status = &ms.AuthorizeEmailAttach(&emailid, "MCFUser", 2, 1);
```

DeleteEmail

Syntax

DeleteEmail*(email ID,forced delete)*

Description

Deletes the specified email from the PeopleSoft database and corresponding attachments from the directory where they are stored.

Parameters

Parameter	***Description***
email ID	The ID used to uniquely identify each email within the PeopleSoft database.

Parameter	Description
forced delete	This parameter is used to set the forced delete flag. It is a Boolean value. A forced delete ensures that the deletion process continues even in cases where the process encounters issues or errors related to an email. For example, with forced delete enabled, the deletion process continues even though the attachments of the email cannot be found or do not exist. True means "force delete" and False means "do not force delete if there is an error in deleting this email."

Returns

Returns a Boolean value. True for success, False otherwise.

Example

```
Local MCFMailStore &ms;

&ms = create MCFMailStore();
&forcedel = True;
&status = &ms.DeleteEmail(&emailid, &forcedel);
```

RetrieveEmail

Syntax

`RetrieveEmail`*`(email ID)`*

Description

Use RetrieveEmail to read email from the PeopleSoft database.

Parameters

Parameter	Description
email ID	The ID that uniquely identifies an email within the PeopleSoft database.

Returns

Returns a rowset.

Example

```
Local MCFMailStore &ms;
Local Rowset &EMAIL_ROWSET;

&ms = create MCFMailStore();
&EMAIL_ROWSET = &ms.RetrieveEmail(&emailid);
```

StoreEmail

Syntax

`StoreEmail`*`(&rowset,row)`*

Description

The StoreEmail method inserts an email retrieved from the mail server into the PeopleSoft database. PeopleSoft stores the contents or pieces of the email in rowsets and child rowsets.

Parameters

Parameter	***Description***
&rowset	The rowset containing the contents of an email. The rowset should be the rowset used by the MCFGetMail class to retrieve emails from the mail server. The rowset should be an already instantiated rowset. **Note.** The first row of the rowset is empty. The first email in the rowset has a row value of 2.
row	The row number in which the email data is stored. **Note.** The first row is always empty. Row 2 always contains the first email of the rowset.

Returns

Returns a PeopleSoft email ID to act as the unique key for a particular email within the PeopleSoft database.

Returns 0 if an error occurred.

Example

```
Local MCFGetMail &GM;
Local Rowset &rs;
Local number &nemails;
Local string &email_id;

&GM = create MCFGetMail();
&rs = &GM.ReadEmailsWithAttach(&user, &password, &server, &ibnode,
&count);

/* Get the number of emails in the recordset from MCF_NUMROWS field */

&nemails = &rs.GetRow(1).GetRecord(Record.MCFEM_RES_MAIN).GetField
(Field.MCF_NUMROWS).Value;

   If (&nemails > 0) Then

/* First row in the rowset is for header info only. Start writing from
 second row */

      For &i = 1 To &nemails
         &email_id = &ms.StoreEmail(&rs, &i + 1);
         If (&email_id <> 0) Then
            /* Store was successful - email ID returned = &email_id */
         Else
            /* Error in storing email */
         End-If;
      End-For;
   End-If;
```

MCFMailUtil Class

Use the MCFMailUtil class to perform utility operations including encoding and decoding text, validating the email address, validating the email domain name, and determining whether the SMTP email server is available.

MCFMailUtil Class Methods

In this section we discuss the MCFMailUtil class methods. The methods are described in alphabetical order.

DecodeText

Syntax

DecodeText*(TextToDecode,&DecodedText)*

Description

Use the DecodeText method to decode text. If you only want to decode a word, use the DecodeWord property.

The text is decoded based on the values used with the EncodeText method.

The decoded text is placed in the *&DecodedText* parameter.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return each of the substitution strings used to format the error message.

Parameters

Parameter	***Description***
TextToDecode	Specify the text that is to be decoded, as a string.
&DecodedText	Specify a string variable, used to hold the decoded text.

Returns

A Boolean value: true if successful, false otherwise.

See Also

Chapter 23, "Mail Classes," DecodeWord, page 1163 and Chapter 23, "Mail Classes," EncodeText, page 1164

Chapter 23, "Mail Classes," GetErrorMsgParam, page 1167; Chapter 23, "Mail Classes," ErrorDescription, page 1173; Chapter 23, "Mail Classes," ErrorDetails, page 1173; Chapter 23, "Mail Classes," MessageNumber, page 1174 and Chapter 23, "Mail Classes," MessageSetNumber, page 1175

DecodeWord

Syntax

DecodeWord*(WordToDecode,&DecodedWord)*

Description

Use the DecodeWord method to decode a word. If you want to decode a string, use the DecodeText method instead.

The word is decoded based on the values used with the EncodeWord method.

The decoded word is placed in the *&DecodedWord* parameter.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return each of the substitution strings used to format the error message.

Parameters

Parameter	*Description*
WordToDecode	Specify the word that is to be decoded, as a string.
&DecodedWord	Specify a string variable, used to hold the decoded word.

Returns

A Boolean value: true if successful, false otherwise.

See Also

Chapter 23, "Mail Classes," DecodeText, page 1161 and Chapter 23, "Mail Classes," EncodeWord, page 1165

Chapter 23, "Mail Classes," GetErrorMsgParam, page 1167; Chapter 23, "Mail Classes," ErrorDescription, page 1173; Chapter 23, "Mail Classes," ErrorDetails, page 1173; Chapter 23, "Mail Classes," MessageNumber, page 1174 and Chapter 23, "Mail Classes," MessageSetNumber, page 1175

EncodeText

Syntax

```
EncodeText(TextToEncode,charset,EncodingStyle, &EncodedText)
```

Description

Use the EncodeText method to encode text. If you only want to encode a single word, use the EncodeWord method instead.

The encoded text is placed in the *&EncodedText* parameter.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return each of the substitution strings used to format the error message.

Using Encoding Styles

You can specify either "B" or "Q" for the *EncodingStyle* parameter, indicating either Base64 or "Quoted-Printable" content-transfer-encoding, respectively.

PeopleSoft recommends using the "Q" encoding when most of the characters to be encoded are in the ASCII character set; otherwise, you should use the "B" encoding. However, a mail reader that claims to recognize encoded text must be able to accept either encoding for any character set that it supports.

Both Base64 and "Quoted-Printable" content-transfer-encoding are defined by RCF 1521.

See http://www.freesoft.org/CIE/RFC/1522/4.htm.

Parameters

Parameter	***Description***
TextToEncode	Specify the text that is to be encoded, as a string.
charset	Specify the character set to be used for encoding the text, as a string. See *PeopleTools 8.51 PeopleBook: Global Technology*, "Selecting and Configuring Character Sets and Language Input and Output," Character Sets in the PeopleSoft Pure Internet Architecture.
EncodingStyle	Specify the encoding style. Valid values are: • B — for Base64 encoding. • Q — for "Quoted-Printable" content- transfer-encoding.
&EncodedText	Specify a string variable used to hold the encoded text.

Returns

A Boolean value: true if successful, false otherwise.

See Also

Chapter 23, "Mail Classes," DecodeText, page 1161 and Chapter 23, "Mail Classes," EncodeWord, page 1165

Chapter 23, "Mail Classes," GetErrorMsgParam, page 1167; Chapter 23, "Mail Classes," ErrorDescription, page 1173; Chapter 23, "Mail Classes," ErrorDetails, page 1173; Chapter 23, "Mail Classes," MessageNumber, page 1174 and Chapter 23, "Mail Classes," MessageSetNumber, page 1175

EncodeWord

Syntax

```
EncodeWord(WordToEncode,charset,EncodingStyle, &EncodedWord)
```

Description

Use the EncodeWord method to encode a word. If you want to encode more than a single word, use the EncodeText method.

The encoded word is placed in the *&EncodedWord* parameter.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return each of the substitution strings used to format the error message.

Using Encoding Styles

You can specify either "B" or "Q" for the *EncodingStyle* parameter, indicating either Base64 or "Quoted-Printable" content-transfer-encoding, respectively.

PeopleSoft recommends using the "Q" encoding when most of the characters to be encoded are in the ASCII character set; otherwise, you should use the "B" encoding. However, a mail reader that claims to recognize encoded text must be able to accept either encoding for any character set that it supports.

Both Base64 and "Quoted-Printable" content-transfer-encoding are defined by RCF 1521.

See http://www.freesoft.org/CIE/RFC/1522/4.htm.

Parameters

Parameter	***Description***
WordToEncode	Specify the word that is to be encoded, as a string.
charset	Specify the character set to be used for encoding the word, as a string. See *PeopleTools 8.51 PeopleBook: Global Technology*, "Selecting and Configuring Character Sets and Language Input and Output," Character Sets in the PeopleSoft Pure Internet Architecture.
EncodingStyle	Specify the encoding style. Valid values are: • B — for Base64 encoding. • Q — for "Quoted-Printable" content- transfer-encoding.
&EncodedWord	Specify a string variable, used to hold the encoded word.

Returns

A Boolean value: true if successful, false otherwise.

See Also

Chapter 23, "Mail Classes," DecodeWord, page 1163 and Chapter 23, "Mail Classes," EncodeText, page 1164

Chapter 23, "Mail Classes," GetErrorMsgParam, page 1167; Chapter 23, "Mail Classes," ErrorDescription, page 1173; Chapter 23, "Mail Classes," ErrorDetails, page 1173; Chapter 23, "Mail Classes," MessageNumber, page 1174 and Chapter 23, "Mail Classes," MessageSetNumber, page 1175

GetErrorMsgParam

Syntax

`GetErrorMsgParam``(`*`&index`*`)`

Description

Use this method to return each of the substitution strings used in the error message.

Parameters

Parameter	*Description*
&index	Specifies which substitution string to return from the array of substitution parameters.

Returns

A string containing the substitution parameter.

Example

```
Local PT_MCF_MAIL:MCFMailUtil &emailutil = create PT_MCF_MAIL:MCFMailUtil();
For &index = 1 To &emailutil.ErrorMsgParamsCount
   Local String &trace  = "Param" | &index | ": '" | &emailutil.GetErrorMsgParam⇒
(&index) | "'";
End-For;
```

See Also

Chapter 23, "Mail Classes," ErrorMsgParamsCount, page 1174

IsDomainNameValid

Syntax

`IsDomainNameValid`*`(domainname)`*

Description

Use this method to check whether a domain name is valid. The isDomainNameValid method queries your DNS server to check whether the domain name is listed in an "MX", or mailbox exchange, record. Please note that not all domains are properly listed by DNS servers.

Note. The number of retries for domain validation is set by the SMTPDNSTimeoutRetries parameter in psappsrv.cfg. The default number of retries is 1.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return each of the substitution strings used to format the error message.

Parameters

Parameter	***Description***
domainname	Specifies the domain name to be validated, as a string.

Returns

A Boolean value: true if the domain name is valid, false otherwise.

Example

```
Local PT_MCF_MAIL:MCFMailUtil &emailutil = create PT_MCF_MAIL:MCFMailUtil();
&result = &emailutil.IsDomainNameValid("oracle.com");
```


See Also

PeopleTools 8.51 PeopleBook: System and Server Administration, "Setting Application Server Domain Parameters," SMTPDNSTimeoutRetries

Chapter 23, "Mail Classes," GetErrorMsgParam, page 1167; Chapter 23, "Mail Classes," ErrorDescription, page 1173; Chapter 23, "Mail Classes," ErrorDetails, page 1173; Chapter 23, "Mail Classes," MessageNumber, page 1174 and Chapter 23, "Mail Classes," MessageSetNumber, page 1175

IsEmailServerAvailable

Syntax

`IsEmailServerAvailable``(server,port,user,password)`

Description

Use the IsEmailServerAvailable method to check if the email server specified with the SMTP settings in the application server configuration file is available.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return each of the substitution strings used to format the error message.

Parameters

Parameter	***Description***
server	Specify the name of the mail server handling the specified user account, as a string.
port	Specify the port of the mail server handling the specified user account, as an integer.
user	Specify the user name on the mail server, such as "support" or "john_doe", as a string.

Parameter	*Description*
password	Specify the password associated with the specified user name, as a string.

Returns

A Boolean value: true if the server is available, false otherwise.

See Also

PeopleTools 8.51 PeopleBook: System and Server Administration, "Setting Application Server Domain Parameters"

Chapter 23, "Mail Classes," GetErrorMsgParam, page 1167; Chapter 23, "Mail Classes," ErrorDescription, page 1173; Chapter 23, "Mail Classes," ErrorDetails, page 1173; Chapter 23, "Mail Classes," MessageNumber, page 1174 and Chapter 23, "Mail Classes," MessageSetNumber, page 1175

ParseRichTextHtml

Syntax

```
ParseRichTextHtml(richtext)
```

Description

Use this method to split rich text content with images into an array of MCFBodyPart objects. Typically, this method is used with rich text produced by the rich text editor on a long edit box.

The first element in the array is the text of the email message. The remaining elements in the array represent each image in the message. The images are added as inline parts of the email, rather than as attachments, preserving the original formatting of the HTML content.

Prior to sending the email, the images are stored in a temporary directory. The default directory is *PS_SERVDIR*/images. Use the MCFMailUtil class imagesLocation property to specify a different directory location. Images in this temporary directory must be deleted manually after sending the email; these image files are not deleted automatically.

Parameters

Parameter	*Description*
richtext	Rich text with images that needs to be parsed.

Returns

An array of MCFBodyPart objects.

Example

See the example on creating an email from rich text editor output later in this chapter.

See Also

Chapter 23, "Mail Classes," MCFBodyPart Class, page 1116

Chapter 23, "Mail Classes," imagesLocation, page 1174

Chapter 23, "Mail Classes," Creating an Email from Rich Text Editor Output, page 1221

ValidateAddress

Syntax

`ValidateAddress``(`*`addresslist`*`)`

In which *addresslist* is a list of email addresses in the form:

`email_address1`` [{,|;} `*`email_address2`*`] ...`

Description

Use the ValidateAddress method to validate a comma- or semicolon-separated list of email addresses. This method checks the syntax of the email address. It does not verify the domain name or the validity of the user name.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- badaddresses
- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return each of the substitution strings used to format the error message.

Parameters

Parameter	*Description*
addresslist	Specify the email address (or addresses) you want to verify as a comma- or semicolon-separated list.

Returns

A Boolean value: true if the specified email addresses are valid, false otherwise. When ValidateAddress returns false, the badaddresses property will contain an array of all invalid addresses.

Example

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFMailUtil &emailutil = create PT_MCF_MAIL:MCFMailUtil();
Local boolean &result = &emailutil.ValidateAddress(&email.Recipients);
Local array of string &bAddresses = &emailutil.badaddresses;

If (&result = False) Then
   If (&bAddresses <> Null) Then
      &i = 0;
      While &bAddresses.Next(&i)
         Warning ("Bad email address in input = " | &bAddresses [&i]);
      End-While;
   End-If;
End-If;
```

See Also

Chapter 23, "Mail Classes," GetErrorMsgParam, page 1167; Chapter 23, "Mail Classes," badaddresses, page 1173; Chapter 23, "Mail Classes," ErrorDescription, page 1173; Chapter 23, "Mail Classes," ErrorDetails, page 1173; Chapter 23, "Mail Classes," MessageNumber, page 1174 and Chapter 23, "Mail Classes," MessageSetNumber, page 1175

MCFMailUtil Class Properties

In this section we discuss the MCFMailUtil class properties. The properties are described in alphabetical order.

badaddresses

Description

When the ValidateAddress method returns false, use the badaddresses property to get an array of invalid email addresses.

This property is effectively read-only.

Note. While this property is actually read-write, from PeopleCode, it is meant to be used in a read-only manner. If this property has a user-specified value, that value will be overwritten automatically when there is an error condition.

See Also

Chapter 23, "Mail Classes," ValidateAddress, page 1171

ErrorDescription

Description

If an error occurs, use the ErrorDescription property to get the description of the error.

This property is effectively read-only.

Note. While this property is actually read-write, from PeopleCode, it is meant to be used in a read-only manner. If this property has a user-specified value, that value will be overwritten automatically when there is an error condition.

See Also

Chapter 23, "Mail Classes," ErrorDetails, page 1173

ErrorDetails

Description

If an error occurs, use the ErrorDetails property to get the details for the error.

This property is effectively read-only.

Note. While this property is actually read-write, from PeopleCode, it is meant to be used in a read-only manner. If this property has a user-specified value, that value will be overwritten automatically when there is an error condition.

See Also

Chapter 23, "Mail Classes," ErrorDescription, page 1173

ErrorMsgParamsCount

Description

Use this property to return the number of substitution parameters as an integer.

This property is read-only.

See Also

Chapter 23, "Mail Classes," GetErrorMsgParam, page 1167

imagesLocation

Description

Use this property to specify a string representing a temporary directory for image storage to be used by the ParseRichTextHtml method. Images in this temporary directory must be deleted manually after sending the email; these image files are not deleted automatically.

If this property is not specified, the default image directory is *PS_SERVDIR*/images.

This property is read-write.

See Also

Chapter 23, "Mail Classes," ParseRichTextHtml, page 1170

Chapter 23, "Mail Classes," Creating an Email from Rich Text Editor Output, page 1221

MessageNumber

Description

This property returns the message number of the error message associated with the failed mail utility method. The message number is associated with messages in the PeopleTools message catalog.

This property is effectively read-only.

Note. While this property is actually read-write, from PeopleCode, it is meant to be used in a read-only manner. If this property has a user-specified value, that value will be overwritten automatically when there is an error condition.

See Also

Chapter 23, "Mail Classes," MessageSetNumber, page 1175

MessageSetNumber

Description

This property returns the message set number of the error message associated with the failed mail utility method. The message set number is associated with messages in the PeopleTools message catalog.

This property is effectively read-only.

Note. While this property is actually read-write, from PeopleCode, it is meant to be used in a read-only manner. If this property has a user-specified value, that value will be overwritten automatically when there is an error condition.

See Also

Chapter 23, "Mail Classes," MessageNumber, page 1174

MCFMultiPart Class

Use the MCFMultiPart class to create complex emails with attachments, HTML content, and so on.

MCFMultiPart Class Methods

In this section, we discuss the MCFMultiPart class methods. The methods are described in alphabetical order.

AddBodyPart

Syntax

`AddBodyPart`*`(&bodyPart)`*

Description

Use the AddBodyPart method to add the specified body part to this multipart email.

Parameters

Parameter	***Description***
&bodyPart	Specify an already instantiated MCFBodyPart object.

Returns

None.

GetBodyPart

Syntax

`GetBodyPart`*`(Index)`*

Description

Use the GetBodyPart method to return a reference to the body part specified by *Index.*

Parameters

Parameter	***Description***
Index	Specify the numeric position of the body part you want to access.

Returns

A reference to an MCFBodyPart object.

See Also

Chapter 23, "Mail Classes," MCFBodyPart Class Methods, page 1116

GetContentType

Syntax

GetContentType()

Description

Use the GetContentType method to determine the content type.

Parameters

None.

Returns

A string.

GetCount

Syntax

GetCount()

Description

Use the GetCount method to return the number of parts in the MCFMultiPart object.

Parameters

None.

Returns

A number.

MCFMultiPart Class Property

In this section we discuss the MCFMultiPart class property.

SubType

Description

Use this property to specify the subtype of the MCFMultiPart object. The values are:

- alternative
- related
- mixed

The default value is mixed.

This property is read-write.

MCFOutboundEmail Class

This class inherits many properties from the MCFEmail class, which in turn inherits many properties from the MCFPart class. Generally, you use only the subclasses, MCFOutboundEmail and MCFInboundEmail, and not the superclasses.

MCFOutboundEmail Class Methods

In this section, we discuss the MCFOutboundEmail class methods. The methods are discussed in alphabetical order.

AddAttachment

Syntax

`AddAttachment``(`*`{FilePath | FileURL}, FilePathType, FileName,FileDescr, OverrideContentType,OverrideCharset`*`)`

Description

Use the AddAttachment method to add an attachment to this email.

Use the *FilePathType* parameter to specify if the file path is relative or absolute. If it's an absolute path, the file path could be a file on the local machine, a network folder, or a fully-qualified URL.

You can also add an attachment by creating a multipart object, adding the attachments as bodyparts to the multipart object, and then setting the MultiPart property of the MCFOutbounEmail object. The AddAttachment method is provided for convenience.

Parameters

Parameter	*Description*
FilePath | *FileURL*	You can either specify the file path to the file, or a URL to the file. Depending on the *FilePathType* parameter, you may specify either a relative or absolute file path to the file. This parameter should include the file name and extension of the file. If you specify a URL address where the file is located, it must be an absolute URL. This URL should include the actual name of the file. If you specify a URL address, you must specify the *FilePathType* parameter as %FilePath_Absolute.
FilePathType	Specify the path type for the file, whether it is an absolute or relative path. The values are: • %FilePath_Absolute — the file path is an absolute path. • %FilePath_Relative — the file path is a relative path.
FileName	Specify the name of the file that you want to attach, as a string. You must include the file extension as well.
Description	Specify a description of the file.
OverrideContentType	The system detects the content type of the attachment using the file location and name. Specifying *OverrideContentType* overrides the system detected content type. The character set can also be included in the ContentType. For example, "text/plain; charset=US-ASCII"
OverrideCharset	Specify a character set to be used to override the existing character set.

Returns

None.

AddHeader

Syntax

`AddHeader`*`(HeaderName,HeaderValue)`*

Description

Use the AddHeader method to add a header to the email. This method allows for email server customizations. Some commonly used headers are already exposed as properties, such as From, To, Subject, ContentType and ContentLanguage. Advanced applications can adapt this technique to meet their own requirements. These headers can be either standard SMTP headers or custom headers starting with "X-".

Parameters

Parameter	***Description***
HeaderName	Specify the name of the header that you want to add. This parameter takes a string value.
HeaderValue	Specify the actual text of the header, as a string.

Returns

None.

Example

```
   Local PT_MCF_MAIL:MCFOutboundEmail &email = create PT_MCF_MAIL:MCFOutboundEmail⇒
();
   Local string &TestName = "Custom Header";

   &email.From = &def.From;
   &email.Recipients = &def.Recipients;
   &email.SMTPServer = &def.SMTPServer;
   &email.Subject = &TestName;
   &email.Text = &TestName;

   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP2");
   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP3");
   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP4");

   &res = &email.Send();
```

See Also

Chapter 23, "Mail Classes," GetHeader, page 1181; Chapter 23, "Mail Classes," GetHeaderCount, page 1182; Chapter 23, "Mail Classes," GetHeaderName, page 1183; Chapter 23, "Mail Classes," GetHeaderNames, page 1183 and Chapter 23, "Mail Classes," GetHeaderValues, page 1184

GetErrorMsgParam

Syntax

GetErrorMsgParam_(&index)_

Description

Use this method to return each of the substitution strings used in the error message.

Parameters

Parameter	*Description*
&index	Specifies which substitution string to return from the array of substitution parameters.

Returns

A string containing the substitution parameter.

Example

```
Local PT_MCF_MAIL:MCFOutboundEmail &email = create PT_MCF_MAIL:MCFOutboundEmail();
For &index = 1 To &email.ErrorMsgParamsCount
   Local String &trace  = "Param" | &index | ": '" | &email.GetErrorMsgParam⇒
(&index) | "'";
End-For;
```

See Also

Chapter 23, "Mail Classes," ErrorMsgParamsCount, page 1174

GetHeader

Syntax

GetHeader_(HeaderName)_

Description

Use the GetHeader method to return the value of the specified header.

Parameters

Parameter	*Description*
HeaderName	Specify the name of the header that you want to access the values for, as a string. The matching of header names is case insensitive.

Returns

An array of string containing the header values. If the header is not present, returns an array of length zero.

See Also

Chapter 8, "Array Class," page 257

Chapter 23, "Mail Classes," AddHeader, page 1179; Chapter 23, "Mail Classes," GetHeaderCount, page 1182; Chapter 23, "Mail Classes," GetHeaderName, page 1183; Chapter 23, "Mail Classes," GetHeaderNames, page 1183 and Chapter 23, "Mail Classes," GetHeaderValues, page 1184

GetHeaderCount

Syntax

GetHeaderCount()

Description

Use the GetHeaderCount method to return the number of headers.

Parameters

None.

Returns

An integer, representing the number of headers present.

See Also

Chapter 23, "Mail Classes," AddHeader, page 1179; Chapter 23, "Mail Classes," GetHeader, page 1181; Chapter 23, "Mail Classes," GetHeaderName, page 1183; Chapter 23, "Mail Classes," GetHeaderNames, page 1183 and Chapter 23, "Mail Classes," GetHeaderValues, page 1184

GetHeaderName

Syntax

`GetHeaderName`*`(Index)`*

Description

Use the GetHeaderName to return the name of the header that is located at *Index.*

Parameters

Parameter	*Description*
Index	Specify the numeric position of the header you want to access.

Returns

A string.

See Also

Chapter 23, "Mail Classes," AddHeader, page 1179; Chapter 23, "Mail Classes," GetHeader, page 1181; Chapter 23, "Mail Classes," GetHeaderCount, page 1182; Chapter 23, "Mail Classes," GetHeaderNames, page 1183 and Chapter 23, "Mail Classes," GetHeaderValues, page 1184

GetHeaderNames

Syntax

`GetHeaderNames``()`

Description

Use the GetHeaderNames method to return an array containing the names of all the headers.

Parameters

None.

Returns

An array of string.

See Also

Chapter 23, "Mail Classes," AddHeader, page 1179; Chapter 23, "Mail Classes," GetHeader, page 1181; Chapter 23, "Mail Classes," GetHeaderCount, page 1182; Chapter 23, "Mail Classes," GetHeaderName, page 1183 and Chapter 23, "Mail Classes," GetHeaderValues, page 1184

GetHeaderValues

Syntax

GetHeaderValues*(Index)*

Description

Use the GetHeaderValues method to return the value for the header located at the position specified by *Index.*

Parameters

Parameter	*Description*
Index	Specify the numeric position of the header whose value you want to access.

Returns

An array of string.

See Also

Chapter 23, "Mail Classes," AddHeader, page 1179; Chapter 23, "Mail Classes," GetHeader, page 1181; Chapter 23, "Mail Classes," GetHeaderCount, page 1182; Chapter 23, "Mail Classes," GetHeaderName, page 1183 and Chapter 23, "Mail Classes," GetHeaderNames, page 1183

Send

Syntax

Send()

Description

Use the Send method to send the email.

When you call the Send method, the following properties might be set based on the type of error:

- InvalidAddresses
- MessageNumber (from the PeopleTools Message Catalog)
- MessageSetNumber (from the PeopleTools Message Catalog)
- ValidSentAddresses
- ValidUnsentAddresses

In addition, a list of substitution strings for the error message can be accessed bythe ErrorMsgParamsCount property and the GetErrorMsgParam method.

Parameters

None.

Returns

A number indicating the status of the send. Values are:

Numeric Value	*Constant Value*	*Description*
0	%ObEmail_ FailedBeforeSending	Email failed before being sent.
1	%ObEmail_Delivered	Email was delivered.
2	%ObEmail_NotDelivered	Email delivery not attempted.
3	%ObEmail_PartiallyDelivered	Email has only been partially delivered. Only some of the addresses in the list of addresses were delivered to.
-1	%ObEmail_ SentButResultUnknown	Email was sent but whether it was successful or not is not known.

See Also

Chapter 23, "Mail Classes," InvalidAddresses, page 1194; Chapter 23, "Mail Classes," ValidSentAddresses, page 1204 and Chapter 23, "Mail Classes," ValidUnsentAddresses, page 1205

Chapter 23, "Mail Classes," MessageNumber, page 1196 and Chapter 23, "Mail Classes," MessageSetNumber, page 1196

SetSMTPParam

Syntax

`SetSMTPParam``(ParamName,ParamValue)`

Description

Use the SetSMTPParam method to set parameters for the SMTP session to be used for sending the email. If you are only sending out one email, use this method. If you are sending many emails, use the SMTPSession class and set the parameters once for all the emails you are sending out.

Note. You should use this method before you use the Send method.

Parameters

Parameter	***Description***
ParamName	Specify the name of the parameter you want to overwrite.
ParamValue	Specify the value for the named parameter that you want used instead of the existing value.

Returns

None.

Example

```
&email.setSMTPParam("mail.mime.decodetext.strict", "false");
&email.setSMTPParam("mail.mime.address.strict", "false");
&email.setSMTPParam("mail.debug, "true");
```

See Also

Chapter 23, "Mail Classes," SMTPSession Class, page 1205

MCFOutboundEmail Class Properties

In this section we discuss the MCFOutboundEmail class properties. The properties are described in alphabetical order.

BackupSMTPPort

Description

Use this property to specify the port number of the backup SMTP server. This is an optional property when creating an email. If you don't specify a value, the default value is 25.

This property is read-write.

BackupSMTPServer

Description

Use this property to specify the server name of the backup SMTP server. The system tries to connect to the backup server if the primary SMTPserver is not available.

This property is read-write.

BackupSMTPSSLClientCertAlias

Description

Use this property to specify the alias for the certificate for Secure Sockets Layer (SSL) client authentication on the backup SMTP server.

This property is read-write.

See Also

Chapter 23, "Mail Classes," BackupSMTPSSLPort, page 1187 and Chapter 23, "Mail Classes," BackupSMTPUseSSL, page 1188

BackupSMTPSSLPort

Description

Use this property to specify the SSL port number for the backup SMTP server. This is an optional property. If you don't specify a value for this property, the default value is 465.

This property is read-write.

See Also

Chapter 23, "Mail Classes," BackupSMTPSSLClientCertAlias, page 1187 and Chapter 23, "Mail Classes," BackupSMTPUseSSL, page 1188

BackupSMTPUserName

Description

Use this property to specify the user name used to sign onto the backup SMTP server. This user name is used for authentication when sending mail using the backup mail server.

This property is read-write.

BackupSMTPUserPassword

Description

Use this property to specify the user password used for signing onto the backup SMTP server. The BackupSMTPUserName and password are used for the authentication when sending mail using the backup mail server.

This property is read-write.

BackupSMTPUseSSL

Description

Use this property to indicate whether the connection to the backup SMTP server will be attempted using SSL or not.

This property takes a Boolean value: true if an SSL connection is to be attempted, false if a non-SSL connection is to be attempted.

This property is read-write.

See Also

Chapter 23, "Mail Classes," BackupSMTPSSLClientCertAlias, page 1187 and Chapter 23, "Mail Classes," BackupSMTPSSLPort, page 1187

BCC

Description

Use this property to specify comma-separated list of addresses of all the blind carbon copy recipients of this email. (A blind carbon copy is a copy of the email that contains all of the text of the email, but does not show any of the other email addresses to which this email was sent.)

This property is read-write.

See Also

Chapter 23, "Mail Classes," CC, page 1189

BounceTo

Description

Use this property to specify the email address the system should direct all bounced mail to.

This property is read-write.

CC

Description

Use this property to specify a comma-separated list of addresses that copies of this email are to be sent to (carbon copy.)

This property is read-write.

See Also

Chapter 23, "Mail Classes," BCC, page 1189

Charset

Description

Use this property to specify the character set for the text or the attachment.

You can also specify this property using the SetAttachmentContent method, or the ContentType property.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SetAttachmentContent, page 1122

Chapter 23, "Mail Classes," ContentType, page 1190

ContentLanguage

Description

Use this property to set the language of this email. More than one language can be set in a comma-separated list. Use the standard abbreviated language names, such as en, fr, de, da, and so on.

You can also include the country codes when you specify the language, such as en-us.

This property is read-write.

See Also

http://www.w3.org/WAI/ER/IG/ert/iso639.htm

http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

ContentType

Description

Use this property to specify the content type of this body part.

You can also specify the content type with the SetAttachmentConent method.

You can also use this property to specify the character set. For example, "text/plain; charset=US-ASCII".

This property is read-write.

See Also

Chapter 23, "Mail Classes," SetAttachmentContent, page 1122

Chapter 23, "Mail Classes," Charset, page 1189

DefaultCharSet

Description

Use this property to specify the character set to be applied to all the parts of the email.

This property is read-write.

Description

Description

Use this property to specify a description of the file attachment associated with this email. If there is no file attachment, this property is ignored.

This property is read-write.

See Also

Chapter 23, "Mail Classes," AddAttachment, page 1178

Disposition

Description

Use this property to specify how the body part is presented in the received mail.

Some email clients ignore the setting of this property and present body parts either inline or as file attachments.

Values for this property are:

Value	*Description*
Attachment	The body part is shown as a file attachment.
Inline	The body part is shown in the body itself.

This property is read-write.

ErrorDescription

Description

If an error occurs, use the ErrorDescription property to get the description of the error.

This property is effectively read-only.

Note. While this property is actually read-write, from PeopleCode, it is meant to be used in a read-only manner. If this property has a user-specified value, that value will be overwritten automatically when there is an error condition.

See Also

Chapter 23, "Mail Classes," ErrorDetails, page 1192

ErrorDetails

Description

If an error occurs, use the ErrorDetails property to get the details of the error.

This property is effectively read-only.

Note. While this property is actually read-write, from PeopleCode, it is meant to be used in a read-only manner. If this property has a user-specified value, that value will be overwritten automatically when there is an error condition.

See Also

Chapter 23, "Mail Classes," ErrorDescription, page 1192

ErrorMsgParamsCount

Description

Use this property to the number of substitution parameters as an integer.

This property is read-only.

See Also

Chapter 23, "Mail Classes," GetErrorMsgParam, page 1181

Filename

Description

If you are adding an attachment to the email, you can specify the name of the file using this property.

PeopleSoft recommends that you keep the file extension specified with this property the same as the original extension found in the file path, otherwise the client applications may not be able to display it properly.

This property is read-write.

FilePath

Description

Specify the path for the file that contains the contents of this email.

Whether this is a relative or absolute path depends on the FilePathType property.

You can also specify a URL to the file using this property, if the FilePathType property is specified as %FilePath_Absolute.

If you specify a value for this property, the 'Text' content is ignored (when used with the ContentType property.)

This property is read-write.

See Also

Chapter 23, "Mail Classes," FilePathType, page 1193

FilePathType

Description

Use this property to specify whether the path specified with the FilePath property is a relative or absolute path. The values for this property are:

Value	***Description***
%FilePath_Relative	The file path specified with the FilePath property is a relative path.
%FilePath_Absolute	The file path specified with the FilePath property is either an absolute path to a file, or a URL to a file.

If you specify a relative path, the file must be available in the FILES folder of application server folder.

If you specify an absolute path, the file could be on the local machine, in any network folder, or a URL.

If you specify a value for this property, the Text property is ignored.

This property is read-write.

From

Description

Use this property to specify the email address of the person sending the email.

You can specify more than one address as from in a comma-separated list.

This property is read-write.

See Also

Chapter 23, "Mail Classes," Recipients, page 1197; Chapter 23, "Mail Classes," ReplyTo, page 1198 and Chapter 23, "Mail Classes," Sender, page 1199

Importance

Description

Use this property to specify the value of the importance header field.

The importance can be set to the following:

- low
- normal
- high

If the Priority property is not set, the system automatically sets it to the corresponding values like 5, 3 and 1.

This property is read-write.

InvalidAddresses

Description

Use this property to get a partial list of email addresses to which the email could not be sent. The email addresses are in a string, separated by commas.

This property is read-only.

See Also

Chapter 23, "Mail Classes," ValidSentAddresses, page 1204

Chapter 23, "Mail Classes," ValidUnsentAddresses, page 1205

IsAuthenticationReqd

Description

Use this property to specify if authentication is required or not. If the SMTP server does not support authentication or authentication is not enabled, this property is ignored.

This property takes a Boolean value: true if authentication is required, false otherwise.

This property is read-write.

IsOkToSendPartial

Description

Use this property to specify if the email can be sent to only some of the specified recipients or if it must be sent to all. This property takes a Boolean value: true if a partial list of addresses is acceptable, false otherwise. If the email must go out to all those listed, or to none at all, specify false.

This property is read-write.

Considerations Using IsOkToSendPartial

If the syntax of the email address is wrong, no mail is sent, regardless of the value of this property. The error is reported.

If the syntax of the email addresses is correct but the email server is unable to send mail to some addresses, the value of this property is used to either send or not send to the partial list of email addresses.

If the syntax of all the email addresses are correct and the email server is able to dispatch mail to all the addresses, no error is reported immediately. Later, if some addresses are found invalid by the recipient email server or gateway, the mail is sent back. In this case the source email server has no way to find the invalidity of the email address in advance.

IsReturnReceiptReqd

Description

Use this property to specify if the receiver must send acknowledgement that the email was received when it's received. This property takes a Boolean value: true if the return receipt is required, false otherwise.

This property is read-write.

MessageNumber

Description

This property returns the message number of the error message associated with this email. The message number is associated with messages in the PeopleTools Message Catalog.

This property is effectively read-only.

Note. While this property is actually read-write, from PeopleCode, it is meant to be used in a read-only manner. If this property has a user-specified value, that value will be overwritten automatically when there is an error condition.

See Also

Chapter 23, "Mail Classes," MessageSetNumber, page 1196

MessageSetNumber

Description

This property returns the message set number of the error message associated with the email.

This property is effectively read-only.

Note. While this property is actually read-write, from PeopleCode, it is meant to be used in a read-only manner. If this property has a user-specified value, that value will be overwritten automatically when there is an error condition.

See Also

Chapter 23, "Mail Classes," MessageNumber, page 1196

MultiPart

Description

The email can contain simple text, one attachment, or a MultiPart object.

If you have assigned a MultiPart object using this property, the text and attachment related properties are ignored.

This property is read-write.

Example

```
Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
&mp.SubType = "alternative; differences=Content-type";

&mp.AddBodyPart(&text);
&mp.AddBodyPart(&html);

&email.MultiPart = &mp;
```

Priority

Description

Use this property to specify the priority of this email.

The values are:

1. Highest Priority
2. High Priority
3. Normal
4. Low Priority
5. Lowest Priority

The default value is 3, that is, normal priority.

This value set with this property is used to set to a header X-Priority field, as this is the most common but non-standard field to show priority. In addition, the corresponding values are set in two header fields: X-MSMail-Priority and Priority. The headers X-Priority or X-MSMail-Priority are set to the corresponding value only if the user does not specify a value.

This property is read-write.

Recipients

Description

Use this property to specify the email addresses of all the main recipients to whom the email is being sent. All the addresses are specified in a comma-separated string.

This property is read-write.

RefIDs

Description

Use this property to specify a value for the REFERENCES header field of the message.

This property is read-write.

ReplyIDs

Description

Use this property to specify a value for the IN-REPLY-TO header field of the message.

This property is read-write.

ReplyTo

Description

Use this property to specify the email address where the reply should be sent. You do not need to specify a value for this property if the value is the same as the From property.

This property is read-write.

See Also

Chapter 23, "Mail Classes," From, page 1194

ResultOfSend

Description

This property returns the result of the Send method. The values are:

Numeric Value	***Constant Value***	***Description***
0	%ObEmail_ FailedBeforeSending	Email failed before being sent.
1	%ObEmail_Delivered	Email was delivered.
2	%ObEmail_NotDelivered	Email delivery not attempted.

Numeric Value	Constant Value	Description
3	%ObEmail_PartiallyDelivered	Email has only been partially delivered. Only some of the addresses in the list of addresses were delivered to.
-1	%ObEmail_ SentButResultUnknown	Email was sent but whether it was successful or not is not known.

This property is read-only.

See Also

Chapter 23, "Mail Classes," Send, page 1184

Sender

Description

Use this property to specify the address of the author of the message. You do not need to specify a value for this property if the value for the From property is the same.

This property is read-write.

See Also

Chapter 23, "Mail Classes," From, page 1194

Sensitivity

Description

Use this property to specify the value of the sensitivity header field.

The values are:

- personal
- private
- company-confidential

This property is read-write.

SMTPPort

Description

Use this property to specify the port number of SMTP server. This is an optional property. If you don't specify a value for this property, the default value is 25.

This property is read-write.

SMTPServer

Description

Use this property to specify the SMTP server to be used when sending this email.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPPort, page 1200; Chapter 23, "Mail Classes," SMTPUserName, page 1201; Chapter 23, "Mail Classes," SMTPUserPassword, page 1201 and Chapter 23, "Mail Classes," SMTPSession Class, page 1205

SMTPSSLClientCertAlias

Description

Use this property to specify the alias for the certificate for SSL client authentication on the SMTP server.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPSSLPort, page 1201; Chapter 23, "Mail Classes," SMTPUseSSL, page 1202 and Chapter 23, "Mail Classes," Using an SSL Connection to Send Email, page 1232

SMTPSSLPort

Description

Use this property to specify the SSL port number for the SMTP server. This is an optional property. If you don't specify a value for this property, the default value is 465.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPSSLClientCertAlias, page 1200; Chapter 23, "Mail Classes," SMTPUseSSL, page 1202 and Chapter 23, "Mail Classes," Using an SSL Connection to Send Email, page 1232

SMTPUserName

Description

Use this property to specify the user name to be used for logging into the SMTP server.

You should only use this property if the SMTP server allows authentication. If the SMTP server does not allow authentication, setting this property has no effect.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPPort, page 1200; Chapter 23, "Mail Classes," SMTPServer, page 1200 and Chapter 23, "Mail Classes," SMTPUserPassword, page 1201

SMTPUserPassword

Description

Use this property to specify the password for the SMTP user. This property is used with the SMTPUserName property.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPPort, page 1200; Chapter 23, "Mail Classes," SMTPServer, page 1200 and Chapter 23, "Mail Classes," SMTPUserName, page 1201

SMTPUseSSL

Description

Use this property to indicate whether the connection to the SMTP server will be attempted using SSL or not. If you don't specify a value for this property, the default value is N.

This property takes a Boolean value: true if an SSL connection is to be attempted, false if a non-SSL connection is to be attempted.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPSSLClientCertAlias, page 1200; Chapter 23, "Mail Classes," SMTPSSLPort, page 1201 and Chapter 23, "Mail Classes," Using an SSL Connection to Send Email, page 1232

StatusNotifyOptions

Description

Use this property to specify when the system should notify the sender. The values are separated by commas in a string. The values are:

Value	*Description*
SUCCESS	Send notification if the email is delivered successfully.
FAILURE	Send notification if the email is not delivered successfully.
DELAY	Send notification if delivery of the email is delayed.

This property is read-write.

See Also

Chapter 23, "Mail Classes," StatusNotifyReturn, page 1203 and Chapter 23, "Mail Classes," TimeToWaitForResult, page 1203

StatusNotifyReturn

Description

Use this property to specify what should be returned as the status notification. You can return either the header or the full message. This option is meaningful only if StatusNotifyOptions property is set.

The values are:

Value	***Description***
HDRS	Return only the headers of the email.
FULL	Return the full email message.

This property is read-write.

Subject

Description

Use this property to specify the subject of the email. A subject can only contain 254 characters.

This property is read-write.

Text

Description

Use this property to specify the text for the email.

This property is read-write.

TimeToWaitForResult

Description

Use this property to specify the number of milliseconds to wait for the result of send email process. If the result comes back within this time, the returned value is a positive number. Otherwise, %ObEmail_SentButResultUnknown (or -1) is returned.

This property is read-write.

See Also

Chapter 23, "Mail Classes," StatusNotifyOptions, page 1202 and Chapter 23, "Mail Classes," StatusNotifyReturn, page 1203

UsedDefaultConfig

Description

Use this property to determine whether the default configuration as specified in application server configuration file was used or not.

This property returns a Boolean value: true if the default configuration was used, false if the value specified with the MCFOutboundEmail or the SMTPSession object was used.

This property is read-only.

UsedPrimaryServer

Description

Use this property to determine if the default SMTP server was used, or the backup server.

This property returns a Boolean value: true if the default SMTP server was used, false if the backup server was used.

This property is read-only.

ValidSentAddresses

Description

This property returns the list of addresses that were valid and that the email was sent to. The email addresses are in a string, separated by commas.

This property is read-only.

See Also

Chapter 23, "Mail Classes," InvalidAddresses, page 1194 and Chapter 23, "Mail Classes," ValidUnsentAddresses, page 1205

ValidUnsentAddresses

Description

Use this property to get a list of the email addresses that are valid, yet the system was unable to send to, due to system problems. The email addresses are separated by commas.

This property is read-only.

See Also

Chapter 23, "Mail Classes," InvalidAddresses, page 1194 and Chapter 23, "Mail Classes," ValidSentAddresses, page 1204

SMTPSession Class

Use the SMTPSession class to when you want to send more than one email using the same SMTP session. If you are only sending a single email, use the MCFOutboundEmail class instead.

SMTPSession Class Methods

In this section we describe the SMTPSession class methods. The methods are described in alphabetical order.

Send

Syntax

`Send``(&Email)`

Description

Use the Send method to send an MCFOutboundEmail object as email.

By default, the system sends the email to the SMTP server using the properties set in the application server configuration file. You can specify different setup parameters by supplying values for the SMTPSession object properties, such as Server, BackupServer, and so on.

Parameters

Parameter	*Description*
&Email	Specify an already instantiated MCFOutboundEMail object to be sent.

Returns

A constant indicating the status of the method. The values are:

Value	*Description*
%ObEmail_Delivered	The email was delivered.
%ObEmail_NotDelivered	Email delivery not attempted.
%ObEmail_PartiallyDelivered	The email was only partially delivered, that is, it was only delivered to some of the recipients.
%ObEmail_ FailedBeforeSending	The email wasn't delivered.
%ObEmail_ SentButResultUnknown	The email was sent but the result is unknown. The TimeToWaitForResult value was not sufficient to get the result of the send process.

See Also

Chapter 23, "Mail Classes," SendAll, page 1206

SendAll

Syntax

`SendAll``(&Emails)`

Description

Use the SendAll method to send a number of emails, all using the same SMTP session and session properties.

Parameters

Parameter	*Description*
&Emails	Specify an array of MCFOutboundEmail objects.

Returns

An array of number. Each item in the array indicates the status of an individual email object. Values are:

Value	*Description*
%ObEmail_Delivered	The email was delivered.
%ObEmail_NotDelivered	Email delivery not attempted.
%ObEmail_PartiallyDelivered	The email was only partially delivered, that is, it was only delivered to some of the recipients.
%ObEmail_ FailedBeforeSending	The email wasn't delivered.
%ObEmail_ SentButResultUnknown	The email was sent but the result is unknown. The TimeToWaitForResult value was not sufficient to get the result of the send process.

See Also

Chapter 23, "Mail Classes," Send, page 1205

SetSMTPParam

Syntax

`SetSMTPParam`*`(ParamName,ParamValue)`*

Description

Use the SetSMTPParam method to set additional parameters for the SMTP session.

You must use this method before you use any Send method.

Parameters

Parameter	***Description***
ParamName	Specify the name of the parameter you want to overwrite.
ParamValue	Specify the value for the named parameter that you want used instead of the existing value.

Returns

None.

SMTPSession Class Properties

In this section we describe the SMTPSession class properties. The properties are described in alphabetical order.

BackupPort

Description

Use this property to specify the port number of backup SMTP server. This is an optional property. If you don't specify a value for this property, the default value is 25.

This property is read-write.

BackupServer

Description

Use this property to specify the name of the backup SMTP server. The system tries connecting to the backup server if the primary SMTP server is not available.

This property is read-write.

BackupSSLClientCertAlias

Description

Use this property to specify the alias for the certificate for SSL client authentication on the backup SMTP server.

This property is read-write.

See Also

Chapter 23, "Mail Classes," BackupSSLPort, page 1209 and Chapter 23, "Mail Classes," BackupUseSSL, page 1210

BackupSSLPort

Description

Use this property to specify the SSL port number for the backup SMTP server. This is an optional property. If you don't specify a value for this property, the default value is 465.

This property is read-write.

See Also

Chapter 23, "Mail Classes," BackupSSLClientCertAlias, page 1209 and Chapter 23, "Mail Classes," BackupUseSSL, page 1210

BackupUserName

Description

Use this property to specify the user name used to sign onto the backup SMTP server.

This user name is used for authentication when sending mail using the backup mail server.

This property is read-write.

BackupUserPassword

Description

Use this property to specify the user password used to sign onto the backup SMTP server.

This user name and password are used for authentication when sending mail using the backup mail server.

This property is read-write.

BackupUseSSL

Description

Use this property to indicate whether the connection to the backup SMTP server will be attempted using SSL or not.

This property takes a Boolean value: true if an SSL connection is to be attempted, false if a non-SSL connection is to be attempted.

This property is read-write.

See Also

Chapter 23, "Mail Classes," BackupSSLClientCertAlias, page 1209 and Chapter 23, "Mail Classes," BackupSSLPort, page 1209

IsAuthenticationReqd

Description

Use this property to specify if authentication is required or not. If the SMTP server does not support authentication or authentication is not enabled, this property is ignored.

This property takes a Boolean value: true if authentication is required, false otherwise.

This property is read-write.

Port

Description

Use this property to specify the SMTP port number to be used for sending this email. This property takes a numeric value. This property is optional. If you don't specify a value for this property, the default value of 25 is used.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPServer, page 1200; Chapter 23, "Mail Classes," SMTPUserName, page 1201; Chapter 23, "Mail Classes," SMTPUserPassword, page 1201 and Chapter 23, "Mail Classes," SMTPSession Class, page 1205

Server

Description

Use this property to specify the name of the SMTP server to be used when sending this email.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPPort, page 1200; Chapter 23, "Mail Classes," SMTPUserName, page 1201; Chapter 23, "Mail Classes," SMTPUserPassword, page 1201 and Chapter 23, "Mail Classes," SMTPSession Class, page 1205

SSLClientCertAlias

Description

Use this property to specify the alias for the certificate for SSL client authentication on the SMTP server.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SSLPort, page 1212 and Chapter 23, "Mail Classes," UseSSL, page 1213

SSLPort

Description

Use this property to specify the SSL port number for the SMTP server. This is an optional property. If you don't specify a value for this property, the default value is 465.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SSLClientCertAlias, page 1211 and Chapter 23, "Mail Classes," UseSSL, page 1213

UsedDefaultConfig

Description

Use this property to determine whether the default configuration as specified in application server configuration file was used or not.

This property returns a Boolean value: true if the default configuration was used, false if the value specified with the MCFOutboundEmail or the SMTPSession object was used.

This property is read-only.

UsedPrimaryServer

Description

Use this property to determine if the default SMTP server was used, or the backup server.

This property returns a Boolean value: true if the default SMTP server was used, false if the backup server was used.

This property is read-only.

UserName

Description

Use this property to specify the user name to be used for logging into the SMTP server.

You should only use this property if the SMTP server allows authentication. If the SMTP server does not allow authentication, setting this property has no effect.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPPort, page 1200; Chapter 23, "Mail Classes," SMTPServer, page 1200 and Chapter 23, "Mail Classes," SMTPUserPassword, page 1201

UserPassword

Description

Use this property to specify the password for the SMTP user. This property is used with the SMTPUserName property.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SMTPPort, page 1200; Chapter 23, "Mail Classes," SMTPServer, page 1200 and Chapter 23, "Mail Classes," SMTPUserName, page 1201

UseSSL

Description

Use this property to indicate whether the connection to the SMTP server will be attempted using SSL or not. If you don't specify a value for this property, the default value is N.

This property takes a Boolean value: true if an SSL connection is to be attempted, false if a non-SSL connection is to be attempted.

This property is read-write.

See Also

Chapter 23, "Mail Classes," SSLClientCertAlias, page 1211 and Chapter 23, "Mail Classes," SSLPort, page 1212

Mail Classes Examples

This section provides examples of the following:

- Creating a text email.
- Creating email and overriding SMTP settings.
- Creating an HTML email.
- Creating a multipart email with text and HTML parts.
- Creating an HTML email with images.
- Creating an email from rich text editor output.
- Creating an email with attachments.
- Creating email attachments by specifying URLs.
- Creating and sending multiple emails.
- Authenticating email while sending.
- Using an SSL connection to send email.

Creating a Text Email

The following code example creates and sends a very simple email, then tests the results.

```
import PT_MCF_MAIL:*;

/*-- Create an outbound email object --*/

Local PT_MCF_MAIL:MCFOutboundEmail &eMail =
create PT_MCF_MAIL:MCFOutboundEmail();

/*-- Initialize the usual fields of an email --*/

&eMail.From = &FromAddress;
&eMail.Recipients = &ToList;
&eMail.Subject = &Subject;
&eMail.Text = &MailBody;

/*-- The send method uses the default SMTP parameters as set in the app server⇒
 configuration file.

This send method makes a connection to the SMTP server, sends the mail and then⇒
 disconnects.

The result are returned as a number corresponding to the possible values.

The list of ValidSent, InvalidSent and Invalid addresses are returned in the email⇒
 object itself
----*/

Local integer &resp = &eMail.Send();
Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating Email and Overriding SMTP Settings

The following code example creates a text email and overrides the SMTP settings as found in the application server configuration file.

```
import PT_MCF_MAIL:*;

/*-- Create an email object by setting individual parameters ---*/

Local PT_MCF_MAIL:MCFOutboundEmail &eMail =
create PT_MCF_MAIL:MCFOutboundEmail();

&eMail.Recipients = &ToList; /* comma separated list of email addresses */
&eMail.CC = &CCList; /* comma separated list of email addresses */
&eMail.BCC = &BCCList; /* comma separated list of email addresses */
&eMail.From = &FromAddress; /* from email address */
&eMail.ReplyTo = &ReplyToAddress; /* in case the reply is to be sent to a⇒
 different email address */
&eMail.Sender = &SenderAddress; /* If different from the "from" address */

&eMail.Subject = &Subject; /* email subject line */
&eMail.Text = &MailBody; /* email body text */


/*-- Override the default SMTP parameters specified in app server configuration⇒
 file ----*/

&eMail.SMTPServer = "psp-smtpg-01";
&eMail.SMTPPort = 10266; /*-- Usually this is 25 by default */

Local integer &resp = &eMail.Send();

/* Now check the &resp for the result */

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating an HTML Email

The following example creates an HTML email.


```
import PT_MCF_MAIL:*;

/*-- Create an email object by setting individual parameters
---*/


Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

   &email.From = &FromAddress;
   &email.Recipients = &ToList;
   &email.Subject = &Subject;

&email.Text =
"<html><body><H1><b>Hi there!</b></H1><P>We are ready.</body></html>";

   &email.ContentType = "text/html";

Local integer &res = &email.Send();

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating a Multipart Email with Text and HTML Parts

The following code example creates an email with both HTML and text sections.

The email client on the target host must be able to detect the HTML and text parts in the email and display the part that the client is configured to display.

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();
Local string &TestName = "Text and its alternate html body";

&email.From = &FromAddress;
&email.Recipients = &ToList;
&email.Subject = &Subject;

Local PT_MCF_MAIL:MCFBodyPart &text = create PT_MCF_MAIL:MCFBodyPart();
&text.Text = "Hi There";

Local PT_MCF_MAIL:MCFBodyPart &html = create PT_MCF_MAIL:MCFBodyPart();
&html.Text =
"<html><BODY><H1>EMail test with HTML content</H1><b>Hi There</b>" |
"<A href='http://www.peoplesoft.com'>Check this out!</A>" |
"<P></BODY></html>";
&html.ContentType = "text/html";

Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
&mp.SubType = "alternative; differences=Content-type";

&mp.AddBodyPart(&text);
&mp.AddBodyPart(&html);

&email.MultiPart = &mp;

Local integer &res = &email.Send();

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```


Creating an HTML Email with Images

The following code example creates an HTML email with embedded images. The content ID for an image can be a reference to a part in the email. In the following code example, the images become a part of the email and are transmitted as attachments to the email.

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

   &email.From = &FromAddress;
   &email.Recipients = &ToList;
   &email.Subject = &Subject;


Local string &htmlText =
"<html>" |
"  <head><title></title></head>" |
"  <body>" |
"    <b>A sample jpg</b><br>" |
"    <IMG SRC=cid:23abc@pc27 width=566 height=424><br>" |
"    <b>End of jpg</b>" |
"  </body>" |
"</html>";

/* In this sample, the htmlText is assembled using | operator
only for the readability.
Specifying just one string can be more efficient
*/

   Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
   &mp.SubType = "related";

   Local PT_MCF_MAIL:MCFBodyPart &html = create PT_MCF_MAIL:MCFBodyPart();
   &html.Text = &htmlText;
   &html.ContentType = "text/html";

   Local PT_MCF_MAIL:MCFBodyPart &image = create PT_MCF_MAIL:MCFBodyPart();

&image.SetAttachmentContent("///file:C:/User/Documentum/XML%20Applications/proddoc⇒
/peoplebook_upc/peoplebook_upc.dtd",
 %FilePath_Absolute, "sample.jpg", "This is a sample image!", "", "");

   &image.AddHeader("Content-ID", "23abc@pc27");

   &mp.AddBodyPart(&html);
   &mp.AddBodyPart(&image);

   &email.MultiPart = &mp;

   Local integer &res = &email.Send();

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
```



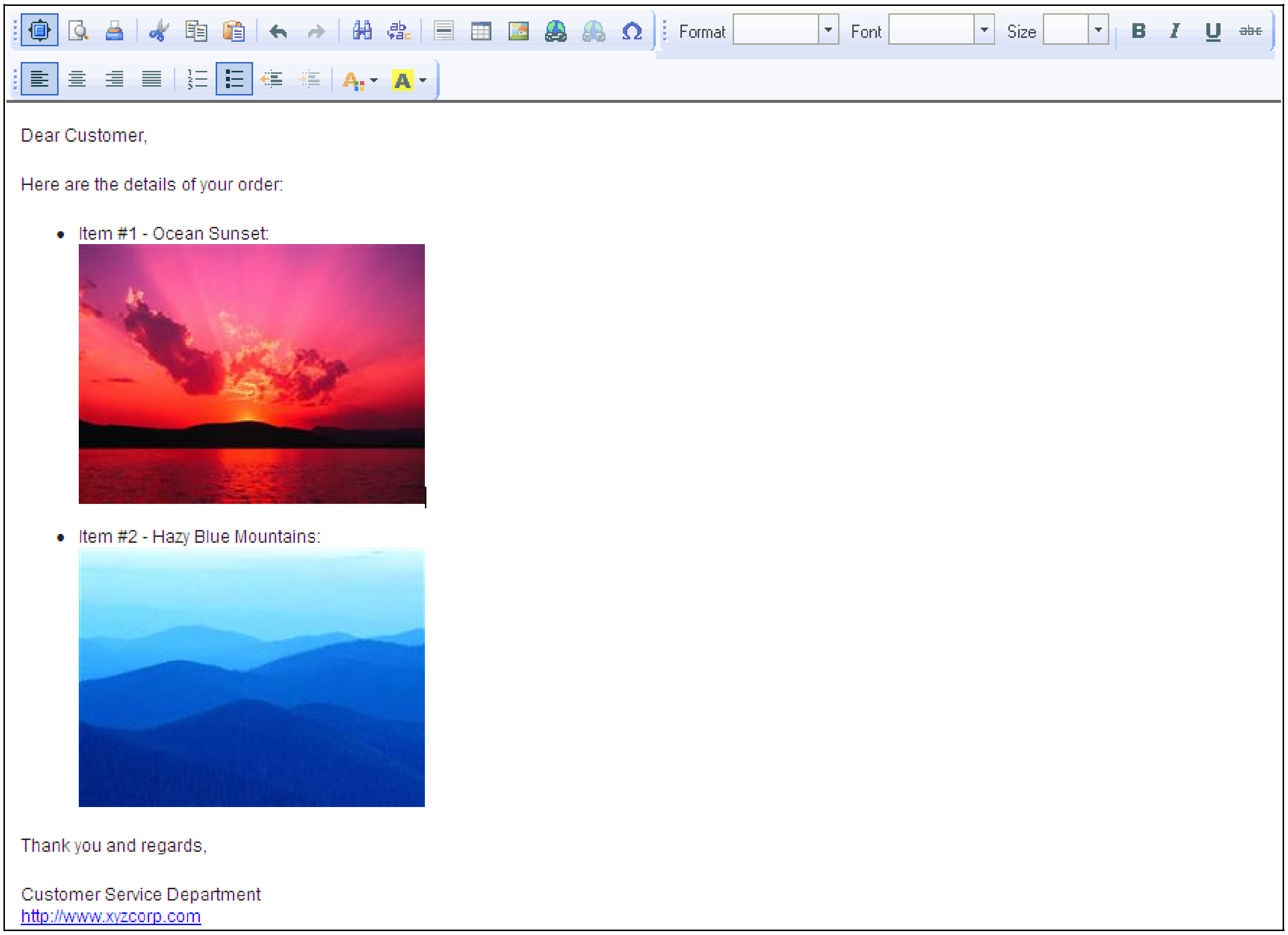
```
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating an Email from Rich Text Editor Output

The following example creates an email message using output from a long edit box enabled with the rich text editor. Using the ParseRichTextHtml method, rich text output of the long edit box is split into an array of MCFBodyPart objects.

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFOutboundEmail &email = create PT_MCF_MAIL:MCFOutboundEmail();
Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
Local PT_MCF_MAIL:MCFMailUtil &mu = create PT_MCF_MAIL:MCFMailUtil();

Local string &htmlText = RECORD_NAME.FIELD_NAME.Value;

&mp.SubType = "related";
&email.Recipients = &recipients;
&email.From = &from;
&email.Subject = "Your Order";

Local array of PT_MCF_MAIL:MCFBodyPart &bp;
&mu.imagesLocation = "c:\temp\images";
&bp = &mu.ParseRichTextHtml(&htmlText);

If (&bp = Null) Then
   MessageBox(0, "", 0, 0, &mu.ErrorDescription);
   Return;
End-If;

&mp.AddBodyPart(&bp [1]);

For &i = 2 To &bp.Len
   &mp.AddBodyPart(&bp [&i]);
End-For;

&email.MultiPart = &mp;
Local integer &resp = &email.Send();

/* Exec code to check the &resp for the result */

/* Exec code to delete temporary image files from the imagesLocation directory */
```

For example, the rich text editor is used to create a message containing text and two images as shown in this example:



Rich text editor displaying an example email

The HTML for this email is as follows:

```
<p>Dear Customer,</p>
<p>Here are the details of your order:</p>
<ul>
<li> Item #1 - Ocean Sunset:
<br /> <img height="180" id="PT_RTE_IMG_DB_LOC###20091111142435187⇒
Sunset_thumbnail.jpg" src="/cs/QEDMO/cache/PTRTDB_JPG_GIYDAOJRGEYT⇒
CMJUGI1DGNJRHA2VG4LOONSXIX2UNB1W1YTOMFUWYLTKOBTQ=_GA==_GE==_2000000000.⇒
JPG" width="240" /><br />  </li>
<li> Item #2 - Hazy Blue Mountains:
<br /> <img height="180" id="PT_RTE_IMG_DB_LOC###20091111142508395⇒
Mountains_thumbnail.jpg" src="/cs/QEDMO/cache/PTRTDB_JPG_GIYDAOJRGEYTCM⇒
JUGI1TAOBTHE1U122VNZ1GC1LOONPXI1DVNVRG3YLJNQXGU3DH_GA==_GE==_2000000000.⇒
JPG" width="240" /></li>
</ul>
<p>Thank you and regards,</p>
<p>Customer Service Department
<br /><a href="http://www.xyzcorp.com">http://www.xyzcorp.com</a></p>
```

When this HTML is parsed by the ParseRichTextHtml method, an array with three MCFBodyPart elements is created. The first MCFBodyPart is array element &bp [1] with the following properties:

- &bp [1].ContentType =

```
text/html
```

- &bp [1].Text =

```
<p>Dear Customer,</p>
<p>Here are the details of your order:</p>
<ul>
<li> Item #1 - Ocean Sunset:
<br /> <img height="180" id="PT_RTE_IMG_DB_LOC###20091111142435187⇒
Sunset_thumbnail.jpg" src="cid:2205@pc27" width="240" />
<br />  </li>
<li> Item #2 - Hazy Blue Mountains:
<br /> <img height="180" id="PT_RTE_IMG_DB_LOC###20091111142508395⇒
Mountains_thumbnail.jpg" src="cid:8160@pc27" width="240" /></li>
</ul>
<p>Thank you and regards,</p>
<p>Customer Service Department
<br /><a href="http://www.xyzcorp.com">http://www.xyzcorp.com</a></p>
```

The second MCFBodyPart is array element &bp [2] with the following properties:

- &bp [2].ContentType =

```
image/gif
```

- &bp [2].Description =

```
Image1
```

- &bp [2].Disposition =

```
inline
```

- &bp [2].AttachmentURL =

```
c:\temp\images/20091111142435187Sunset_thumbnail.jpg
```

The final MCFBodyPart is array element &bp [3] with the following properties:

- &bp [3].ContentType =

```
image/gif
```

- &bp [3].Description =

```
Image2
```

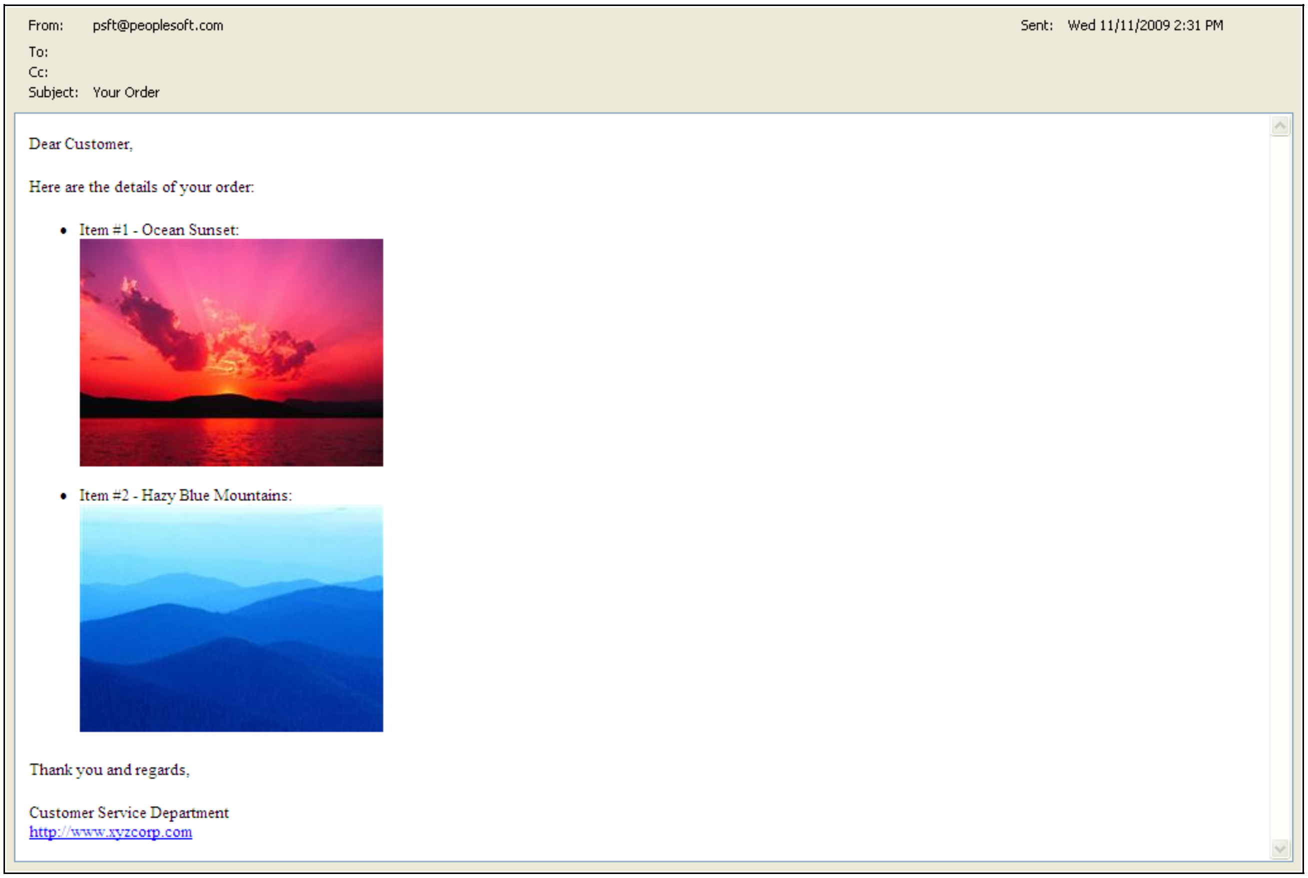
- &bp [3].Disposition =

```
inline
```

- &bp [3].AttachmentURL =

```
c:\temp\images/20091111142508395Mountains_thumbnail.jpg
```

Observe that the email received by the recipient is formatted the way that it appeared in the rich text editor as shown in the following example:



Email as received in Microsoft Outlook

Finally, the XML code that represents this email message is as follows:


```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:⇒
cms="http://cms.ocs.oracle/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/⇒
envelope/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <cms:GetEntitySnapshotResponse>
         <return xsi:type="EmailMessageType_DEBUG" SnapshotId="00000000000000010⇒
00009E2607E2B4200000124E528DD58000000000000000001" CEN="334B:3BF0:emsg:⇒
38893C00F42F38A1E0404498C8A6612B0000FEE32642">
            <CreatedOn>2009-11-11T21:31:07Z</CreatedOn>
            <Creator EID="334B:3BF0:user:CB3F97695BD24114B65149D2E4CC842E0000000⇒
112AE"/>
            <Deleted>false</Deleted>
            <ModifiedBy EID="334B:3BF0:user:CB3F97695BD24114B65149D2E4CC842E0000⇒
000112AE"/>
            <ModifiedOn>2009-11-11T21:31:07Z</ModifiedOn>
            <Name>your order</Name>
            <Parent EID="334B:3BF0:afrh:38893C00F42F38A1E0404498C8A6612B00001EF9⇒
C00A"/>
            <New>false</New>
            <ReadFlag>true</ReadFlag>
            <UserCreatedOn>2009-11-11T21:31:06Z</UserCreatedOn>
            <Viewer EID="334B:3BF0:user:CB3F97695BD24114B65149D2E4CC842E00000001⇒
12AE"/>
            <Content>
               <ContentId>24316678.11257950022359.JavaMail.admin@XXXXX</ContentId>
               <MediaType>message/rfc822</MediaType>
               <Body xsi:type="MultiContentType">
                  <MediaType>multipart/related</MediaType>
                  <Parts>
                     <Item xsi:type="SimpleContentType">
                        <MediaType>text/html</MediaType>
                        <CharacterEncoding>us-ascii</CharacterEncoding>
                        <Language>en</Language>
                        <ContentStream>
                           <Id>T0RNMU5ESTFOVFpEUmtKQ05FWTJNemhETmpWQ1JqWXdRel⇒
ZDT0VJek9EUXdNREF3TURBd01EWTVRVUU9JCQjIyQkN2JpdCQkIyMkJDg3NyQkIyMkJDU2Mg==</Id>
                           <Length>640</Length>
                        </ContentStream>
                     </Item>
                     <Item xsi:type="SimpleContentType">
                        <Disposition>INLINE</Disposition>
                        <ContentId>&lt;2205@pc27&gt;</ContentId>
                        <MediaType>image/gif</MediaType>
                        <CharacterEncoding>ascii</CharacterEncoding>
                        <Language>en</Language>
                        <ContentStream>
                           <Id>T0RNMU5ESTFOVFpEUmtKQ05FWTJNemhETmpWQ1JqWXdRelZD⇒
T0VJek9EUXdNREF3TURBd01EWTVRVUU9JCQjIyQkYmFzZTY0JCQjIyQkMTY3NCQkIyMkJDExNzI4</Id>
                           <Length>11923</Length>
                        </ContentStream>
                        <Name>20091111142435187Sunset_thumbnail.jpg</Name>
                     </Item>
                     <Item xsi:type="SimpleContentType">
                        <Disposition>INLINE</Disposition>
                        <ContentId>&lt;8160@pc27&gt;</ContentId>
                        <MediaType>image/gif</MediaType>
                        <CharacterEncoding>ascii</CharacterEncoding>
                        <Language>en</Language>
                        <ContentStream>
                           <Id>T0RNMU5ESTFOVFpEUmtKQ05FWTJNemhETmpWQ1JqWXdRelZD⇒
```



```
T0VJek9EUXdNREF3TURBd01EWTVRVUU9JCQjIyQkYmFzZTY0JCQjIyQkMTM2NDMkJCMjJCQ3MzUy</Id
>
                           <Length>7553</Length>
                        </ContentStream>
                        <Name>20091111142508395Mountains_thumbnail.jpg</Name>
                     </Item>
                  </Parts>
               </Body>
               <From>
                  <Address>&lt;psft@peoplesoft.com&gt;</Address>
               </From>
               <Priority>NONE</Priority>
               <Sender>
                  <Address>&lt;psft@peoplesoft.com&gt;</Address>
               </Sender>
               <SentDate>2009-11-11T21:31:06Z</SentDate>
               <Size>21039</Size>
               <Subject>Your Order</Subject>
               <TOReceivers>
                  <Item>
                     <Address>&lt;&gt;</Address>
                     <Participant EID="334B:3BF0:user:CB3F97695BD24114B65149D2E4⇒
CC842E0000000112AE"/>
                     <Directives/>
                     <Status>DELIVERED</Status>
                  </Item>
               </TOReceivers>
            </Content>
            <DeliveredTime>
               <DateOnly>false</DateOnly>
               <FloatingTime>false</FloatingTime>
               <Timestamp>2009-11-11T21:31:07Z</Timestamp>
            </DeliveredTime>
            <Modifiable>false</Modifiable>
            <ReceiptRequested>false</ReceiptRequested>
            <Type>EMAIL</Type>
            <MimeHeaders>
               <Item>
                  <Name>X-AUTH-TYPE</Name>
                  <Value>SW50ZXJuYWwgSVA=</Value>
               </Item>
               <Item>
                  <Name>Received</Name>
                  <Value>ZnJvbSByZ21pbmV0MTMub3JhY2xlLmNvbSAoLzE0OC44Ny4xM⇒
TMuMTI1KSBieSBkZWZhdWx0IChPcmFjbGUgQmVlaGl2ZSBHYXRld2F5IHY0LjApIHdpdGggRVNN⇒
VFAgOyBXZWQsIDExIE5vdiAyMDA5IDEzOjMxOjA2IC0wODAw</Value>
               </Item>
               <Item>
                  <Name>Received</Name>
                  <Value>ZnJvbSBCVUZGWSAoYnVmZnkudXMub3JhY2xlLmNvbSBbMTAuMTM4L⇒
jIyOC4xMzNdKSBieSByZ21pbmV0MTMub3JhY2xlLmNvbSAoU3dpdGNoLTMuMy4xL1N3aXRjaC0⇒
zLjMuMSkgd2l0aCBFU01UUCBpZCBuQUJMVjZVSjAxMzk0NiBmb3IgPGRhdmUucGllcmNlQG9yYWNsZ⇒
S5jb20+OyBXZWQsIDExIE5vdiAyMDA5IDIxOjMxOjA2IEdNVA==</Value>
               </Item>
               <Item>
                  <Name>MIME-Version</Name>
                  <Value>MS4w</Value>
               </Item>
               <Item>
                  <Name>To</Name>
                  <Value>ZGF2ZS5waWVyY2VAb3JhY2xlLmNvbQ==</Value>
               </Item>
```



```
                <Item>
                   <Name>MESSAGE-ID</Name>
                   <Value>PDI0MzE2Njc4LjExMjU3OTUwMDIyMzU5LkphdmFNYWlsLmFkb⇒
WluQEJVRkZZPg==</Value>
                </Item>
                <Item>
                   <Name>Content-Type</Name>
                   <Value>bXVsdGlwYXJ0L3JlbGF0ZWQ7ICBib3VuZGFyeT0iLS0tLT1fUGFyd⇒
F8xXzIzNzUyMTguMTI1Nzk1MDAyMjEwOSI=</Value>
                </Item>
                <Item>
                   <Name>From</Name>
                   <Value>cHNmdEBwZW9wbGVzb2Z0LmNvbQ==</Value>
                </Item>
                <Item>
                   <Name>X-SOURCE-IP</Name>
                   <Value>YnVmZnkudXMub3JhY2xlLmNvbSBbMTAuMTM4LjIyOC4xMzNd</Value>
                </Item>
                <Item>
                   <Name>X-CT-REFID</Name>
                   <Value>c3RyPTAwMDEuMEEwOTAyMDQuNEFGQjJEMTkuMDBGQSxzcz0xLGZncz⇒
0w</Value>
                </Item>
                <Item>
                   <Name>Subject</Name>
                   <Value>WW91ciBPcmRlcg==</Value>
                </Item>
                <Item>
                   <Name>Date</Name>
                   <Value>V2VkLCAxMSBOb3YgMjAwOSAyMTozMTowNiBHTVQ=</Value>
                </Item>
             </MimeHeaders>
          </return>
       </cms:GetEntitySnapshotResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Creating an Email with Attachments

The following code example creates an email with an attachment.

PeopleSoft recommends that you always provide the proper extension in the file name, otherwise the receiving email client may not be able to associate it with the appropriate application.

Note. There is no automatic restriction of attachment size using the mail classes. To restrict an email based on attachment size, your application must check the size of an attachment before composing and sending the email.

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

&email.From = &FromAddress;
&email.Recipients = &ToList;
&email.Subject = &Subject;

Local string &plain_text = "Hi there!";
Local PT_MCF_MAIL:MCFBodyPart &text = create PT_MCF_MAIL:MCFBodyPart();
&text.Text = &plain_text;

Local PT_MCF_MAIL:MCFBodyPart &attach1 = create PT_MCF_MAIL:MCFBodyPart();
&attach1.SetAttachmentContent("Ocean Wave.jpg", %FilePath_Relative,
"Ocean Wave.jpg", "Ocean Wave", "", "");
/* %FilePath_Relative indicates the file is available at Appserver's FILES⇒
 dierctory */

Local PT_MCF_MAIL:MCFBodyPart &attach2 = create PT_MCF_MAIL:MCFBodyPart();
&attach2.SetAttachmentContent("///file:C:/User/Documentum/XML%20Applications⇒
/proddoc/peoplebook_upc/peoplebook_upc.dtd",
 %FilePath_Absolute, "Sample.jpg", "Sample", "", "");
/* The Sample.jpg is available in the "public" folder of my-server machine*/


Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
&mp.AddBodyPart(&text);
&mp.AddBodyPart(&attach1);
&mp.AddBodyPart(&attach2);

&email.MultiPart = &mp;

Local integer &res = &email.Send();

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```


Creating Email Attachments by Specifying URLs

The following code example creates an email attachment specifying a URL for the file location, instead of an absolute or relative path to the file.

```
Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

   &email.From = &FromAddress;
   &email.Recipients = &ToList;
   &email.Subject = &Subject;

   Local string &plain_text = "Hi there!";
   Local PT_MCF_MAIL:MCFBodyPart &text = create PT_MCF_MAIL:MCFBodyPart();

   &text.Text = &plain_text;

Local PT_MCF_MAIL:MCFBodyPart &attach1 = create PT_MCF_MAIL:MCFBodyPart();

&attach1.SetAttachmentContent("http://www.yahoo.com/members_agreement",
 %FilePath_Absolute, "hotmail.htm", "Hotmail Home page", "", "");

Local PT_MCF_MAIL:MCFBodyPart &attach2 = create PT_MCF_MAIL:MCFBodyPart();

&attach2.SetAttachmentContent("ftp://www.w3c/docs/somedoc",
 %FilePath_Absolute, "somedoc.htm", "somedoc from w3c", "", "");

   Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
   &mp.AddBodyPart(&text);
   &mp.AddBodyPart(&attach1);
   &mp.AddBodyPart(&attach2);

   &email.MultiPart = &mp;

Local integer &res = &email.Send();

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```


Creating and Sending Multiple Email Messages

You can create several emails and send them all in a single connection to the SMTP server. However, your system administrator may also need to configure the SMTP server with longer connection time-outs if you are sending multiple emails.

```
import PT_MCF_MAIL:*;

/*-- Create an email object by setting individual parameters
---*/
 Local array of PT_MCF_MAIL:MCFOutboundEmail &mails;
 Local PT_MCF_MAIL:MCFOutboundEmail &email;

 Local PT_MCF_MAIL:SMTPSession &commonSession =
create PT_MCF_MAIL:SMTPSession();

   &email = &commonSession.CreateOutboundEmail();
   &email.From = &FromAddress;
   &email.Recipients = &ToList1;
   &email.Subject = &Subject;
   &email.Text = &MailBody1;

   &mails = CreateArray(&email);

   &email = &commonSession.CreateOutboundEmail();
   &email.From = &FromAddress;
   &email.Recipients = &ToList2;
   &email.Subject = &Subject;
   &email.Text = &MailBody2;
   &mails [2] = &email;

   &email = &commonSession.CreateOutboundEmail();
   &email.From = &FromAddress;
   &email.Recipients = &ToList3;
   &email.Subject = &Subject;
   &email.Text = &MailBody3;
   &mails [3] = &email;

   Local array of integer &allRes = &commonSession.SendAll(&mails);
```

Sending an Email With Authentication

The following code example includes a user name and password to be used by the SMTP server.

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

   &email.From = &FromAddress;
   &email.Recipients = &ToList;
   &email.Subject = &Subject;
   &email.Text = &MailBody;

   &email.SMTPServer = "MySMTPServer";
   &email.IsAuthenticationReqd = True;

   &email.SMTPUserName = "SomeLoginName";
   &email.SMTPUserPassword = "SomeLoginPassword";

   Local integer &res = &email.Send();
```

Using an SSL Connection to Send Email

The following code example uses an SSL connection to send an email message:

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFOutboundEmail &email = create PT_MCF_MAIL:MCFOutboundEmail();

&email.From = &FromAddress;
&email.Recipients = &ToList;
&email.Subject = &Subject;
&email.Text = &MailBody;

&email.SMTPServer = "MySMTPServer";
&email.IsAuthenticationReqd = True;

&email.SMTPUserName = "SomeLoginName";
&email.SMTPUserPassword = "SomeLoginPassword";
&email.SMTPUseSSL = "Y";
&email.SMTPSSLClientCertAlias = &cAlias;
&email.SMTPSSLPort = &SSLPort;

Local integer &res = &email.Send();
```


Chapter 24

MCF IM Classes

This chapter provides an overview of the MCFIMInfo class and the MCFIMSingleButton class and discusses:

- MCFIMInfo class
- MCFIMSingleButton class

Understanding the MCFIMInfo Class

This section provides an overview of:

- Using the MCFIMInfo class.
- Data type for MCFIMInfo objects.
- Scope of MCFIMInfo objects.

Use the MCFIMInfo class to launch an instant messaging session from a PeopleSoft page and initiate a chat. A developer can use either the PeopleCode class or a push-button with specific characteristics in Application Designer to initiate an instant messaging session.

Note. PeopleSoft recommends using the Application Designer push-button rather than PeopleCode in most applications. The PeopleCode program determines user presence information from the application server, whereas the push-button determines user presence from the browser. The application server must wait until it has *all* the presence information for that page before it can render the page. Though this processing is multi-threaded, it can still be slower than the push-button.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft MultiChannel Framework, "Configuring Instant Messaging in PeopleSoft MultiChannel Framework"

Using the MCFIMInfo Class

Only use the MCFIMInfo class if your application requires the flexibility the PeopleCode can provide. The only supported networks are AOL and Yahoo. SameTime is supported using the Application Designer push-button, but not in PeopleCode.

Instant messages are sent by using the native instant message clients. This means that you can only use this for Microsoft Windows.

The page from which you launch the instant messaging session must be refreshed in order to update presence status.

Data Type for MCFIMInfo Objects

MCFIMInfo objects are declared as type MCFIMInfo. For example:

```
Local MCFIMInfo &MyChat;
```

Scope of MCFIMInfo Objects

An MCFIMInfo object can be only instantiated from PeopleCode.

Use this object only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message subscription, a Component Interface, and so on.

Understanding the MCFIMSingleButton Class

Use the MCFIMSingleButton class to generate a single presence icon that displays all specified screen names. When a user moves the mouse pointer over a single presence icon, it will open a menu with all specified user screen names and their current online status.

Importing the MCFIMSingleButton Class

The MCFIMSingleButton class is an application classes, not a built-in class, like Rowset, Field, Record, MCFIMInfo, and so on. Before you can use this class in your PeopleCode program, you must import it into your program. An import statement either names a particular application class or imports all the classes in a package. Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are not made available.

To import the MCFIMSingleButton class, use the following import statement:

```
import PT_MCF_IM:*;
```

MCFIMInfo Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateMCFIMInfo

MCFIMInfo Class Methods

In this section, the MCFIMInfo class methods are presented in alphabetical order.

AddUser

Syntax

`AddUser``(`*`User`*`)`

Description

Use the AddUser method to add a user to the instant messaging session.

Parameters

Parameter	***Description***
User	Specify the user that you want to add to the instant messaging session, as a string.

Returns

A Boolean value: True if the user was successfully added, False otherwise.

CheckAll

Syntax

`CheckAll``()`

Description

Use the CheckAll method to check the status of users on the instant messaging session.

This method makes the actual network requests to retrieve presence information.

Parameters

None.

Returns

A Boolean: True if the check is successful, False otherwise.

GetAdditionalUserInfo

Syntax

`GetAdditionalUserInfo``()`

Description

This method returns additional user information.

Parameters

None.

Returns

String.

GetErrorImageName

Syntax

`GetErrorImageName``()`

Description

Use GetErrorImageName to return the name of the image used for errors.

Parameters

None.

Returns

String.

GetOffLineImageName

Syntax

`GetOffLineImageName``()`

Description

Use GetOffLineImageName to return the name of the image used to indicate off line status.

Parameters

None.

Returns

String.

GetOnlineImageName

Syntax

`GetOnlineImageName``()`

Description

Use GetOnlineImageName to return the name of the image used to indicate online status.

Parameters

None.

Returns

String.

GetUnknownImageName

Syntax

`GetUnknownImageName``()`

Description

Use GetUnknownImageName to return the image of unknown users.

Parameters

None.

Returns

String.

GetLaunchURL

Syntax

`GetLaunchURL`*`(User)`*

Description

Use the GetLaunchURL method to return the URL that launches the native client and contacts the user. This is a local URL of one of the following forms:

`aim:. . .`

or

`ymsgr:. . .`

Parameters

Parameter	*Description*
User	Specify the user that you want to contact via the launch URL.

Returns

A string representing the URL to launch the native client.

GetStatus

Syntax

`GetStatus`*`(User)`*

Description

Use the GetStatus method to return the current online status of the user specified by *User*. This is the status at the time of the last call to **CheckAll.**

Parameters

Parameter	***Description***
User	Specify the name of the user for whom you want to check the status of.

Returns

An integer. The values are:

Value	***Description***
-1	Network Disabled
0	User offline
1	User online
2	User Unknown
3	Error

RemoveUser

Syntax

`RemoveUser`*`(User)`*

Description

Use the RemoveUser method to remove a user from the instant messaging session.

Parameters

Parameter	***Description***
User	Specify the name of the user you want to remove from the session, as a string.

Returns

A Boolean value: True if the user was successfully removed, False otherwise.

MCFIMSingleButton Class Methods

In this section, the MCFIMSingleButton class methods are presented in alphabetical order.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft MultiChannel Framework, "Configuring Instant Messaging in PeopleSoft MultiChannel Framework," Developing a Sample Application with Single Presence Button

generateHTML

Syntax

`generateHTML``(&RS,slide_dir,X_pos, Y_pos,width,height,seq_num)`

Description

Use this method to generate the HTML code as a string that is needed to display the single presence IM button on a page.

Parameters

Parameter	***Description***
&RS	Specifies the Rowset object that contains the IM user IDs, protocols, domains, and other IM information in the format of the MCF_IMSCRNNAMES record.

Parameter	*Description*
slide_dir	Specifies the direction of the sliding window as a string. Valid values are: left, right, up, down.
X_pos	Specifies the X coordinate (in pixels) of the sliding window's upper left corner as an Integer.
Y_pos	Specifies the Y coordinate (in pixels) of the sliding window's upper left corner as an Integer.
width	Specifies the minimum width of the sliding window as an Integer (in pixels)
height	Specifies the minimum height of the sliding window as an Integer (in pixels)
seq_num	Specifies a unique sequence number for this single presence button as an Integer. If there is more than one single presence button per page, each needs to be identified with a unique sequence number.

Returns

A string representing the HTML code for the single presence button.

Example

Use the page activate event to place the single presence button on a page:

```
import PT_MCF_IM:*;

Global PT_MCF_IM:MCFIMSingleButton &single;
Local Rowset &RS1;

&single = create PT_MCF_IM:MCFIMSingleButton();
/* Number of milliseconds that we want to show sliding window even after we   */
/* remove cursor from single presence button.                                 */

&single.hideDelay = 5325;
&RS1 = CreateRowset(Record.MCF_IMSCRNNAMES);
/* &RS1 have all screen names, right is direction of sliding window. 190 is   */
/* X coordinate of top left corner of sliding window, 200 is Y coordinate of  */
/* top left corner of sliding window, 350 is height of sliding window, 450    */
/* is width of sliding window,  1 is some unique no of sliding window on that */
/* page. If there are many single presence icons on page, then it would be    */
/* useful for uniqness.                                                       */

RECORDNAME.MCF_IM_HTMLAREA.Value = &single.generateHTML(&RS1, "right", 190, 200,⇒
 350, 450, 1);
RECORDNAME.MCF_XMPPHEADER = &single.generateJavaScript();
```

See Also

Chapter 24, "MCF IM Classes," generateJavaScript, page 1242

generateJavaScript

Syntax

generateJavaScript()

Description

Use this method to generate JavaScript code that will be used to show single button presence. Call this method only after calling generateHTML.

Parameters

None.

Returns

A string representing the JavaScript code.

See Also

Chapter 24, "MCF IM Classes," generateHTML, page 1240

insertXMPPServerUserData

Syntax

insertXMPPServerUserData(*PS_user_ID*,*protocol*,*XMPP_domain*,*IM_user_ID*,*IM_pwd*)

Description

Use this method to insert a row of user data into the PS_MCF_USER_IM_CFG table.

Parameters

None.

Parameter	*Description*
PS_user_ID	Specifies the PeopleSoft user ID as a string.

Parameter	*Description*
protocol	Specifies the XMPP protocol as a string. The only valid value is: XMPP.
XMPP_domain	Specifies the XMPP domain name as a string.
IM_user_ID	Specifies the email user ID on the XMPP server as a string. For example, if the email ID is first.last@company.com, then the *IM_user_ID* parameter must be set to: first.last.
IM_pwd	Specifies the password for the email user ID as a string.

Returns

A Boolean value: True if insert was successful, False otherwise.

MCFIMSingleButton

Syntax

`MCFIMSingleButton``()`

Description

Use this constructor to instantiate a MCFIMSingleButton object.

Parameters

None.

Returns

A MCFIMSingleButton object.

updateXMPPServerUserData

Syntax

`updateXMPPServerUserData`*`(PS_user_ID,protocol,XMPP_domain,IM_user_ID,IM_pwd)`*

Description

Use this method to update a row of user data in the PS_MCF_USER_IM_CFG table.

Parameters

None.

Parameter	*Description*
PS_user_ID	Specifies the PeopleSoft user ID as a string.
protocol	Specifies the XMPP protocol as a string. The only valid value is: XMPP.
XMPP_domain	Specifies the XMPP domain name as a string.
IM_user_ID	Specifies the email user ID on the XMPP server as a string. For example, if the email ID is first.last@company.com, then the *IM_user_ID* parameter must be set to: first.last.
IM_pwd	Specifies the password for the email user ID as a string.

Returns

A Boolean value: True if update was successful, False otherwise.

MCFIMSingleButton Class Properties

In this section, the MCFIMSingleButton class properties are presented in alphabetical order.

hideDelay

Description

Use this property to set or return an Integer value representing a delay in milliseconds. A user removes the mouse pointer from the single presence button, the delay determines the amount of time the menu remains displayed before it is hidden. The default value is 1,325 milliseconds.

This property is read-write.

Chapter 25

Message Classes

This chapter provides an overview of the message classes and discusses the following topics:

- Data types of a message objects.
- Scope of a message objects.
- Message object population.
- Message segments.
- Content-based routing.
- Error handling.
- Message classes reference.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Understanding PeopleSoft Integration Broker"

Understanding Message Classes

You can create the following types of messages using PeopleSoft Pure Internet Architecture::

- *Rowset-based* messages, which use record definitions to create a hierarchical structure for the data. These are generally used for data from applications, pages, components, and so on.
- *Nonrowset-based* messages, which do not use record definitions. These messages can have virtually any content or structure.
- *Container* messages, which are messages made up of one or more part messages.

Use the PeopleCode message classes to instantiate message objects based on existing message definitions, as well as to populate the objects with data and manipulate the data. You can also use PeopleCode to publish a message.

Rowset-based messages are built on top of the rowset, row, record, and field classes, so the PeopleCode written to populate or access those types of messages looks similar to PeopleCode written to populate or access the component buffer.

Nonrowset-based messages contain XML data, and can be accessed using the XmlDoc class methods and properties.

Container messages contain parts. Each part is a separate message. A container message can contain either all rowset-based messages, *or* all nonrowset-based messages.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Messages, Service Operations and Handlers

Message definitions only define the shape of the data contained in a message. A message definition doesn't contain any other type of information, such as if the message is being used synchronously or if it's a response message.

After you create a message definition, you can use it in or more service operations. The service operation contains all the information about how the message is to be used, the routing, the queue if appropriate, and so on.

At least one handler is defined with every service operation. Handlers define additional programming to be used with processing the message associated with the service operation. The handlers can be thought of as corresponding to pre PeopleSoft 8.48 message events.

The PeopleCode for the various handlers is contained in the Integration Broker application classes.

Integration Broker Application Classes

The Integration Broker application classes house the processing logic for asynchronous and synchronous messages. By implementing the Integration Broker application classes, you can reuse code more easily and access the other benefits of using application classes.

The following application classes are defined for Integration Broker. To access these application classes, in PeopleTools Application Designer, open the PS_PT application package, then open the Integration subpackage.

All of the Integration Broker application classes are defined as interfaces. This means that there is no native implementation of them: you must import them to your program and implement them if you want to use them.

Application Class	***Methods Contained in Application Class***	***Comments***
INotificationHandler	OnNotify OnError	This interface is the equivalent of the Subscription event in releases prior to PeopleTools 8.48.

Application Class	***Methods Contained in Application Class***	***Comments***
IPrePostNotification	OnPreNotify OnPostNotify OnError	This interface provides pre-processing and post-processing of segmented messages.
IReceiver	OnAckReceive OnError	This interface is the equivalent of the OnAckReceive Message event in releases prior to PeopleTools 8.48.
IRequestHandler	OnRequest OnError	This interface is the equivalent of the OnRequest Message event in releases prior to PeopleTools 8.48.
IRouter	OnRouteSend OnRouteReceive OnError	This interface is the equivalent of the OnRouteSend and OnRouteReceive Message events in releases prior to PeopleTools 8.48.
ISend	OnRequestSend OnError	This interface is the equivalent of the OnSend Message event in releases prior to PeopleTools 8.48.

Data Types of Message Objects

Every message class is its own data type, that is, messages are declared as data type Message, IntBroker objects are declared as type IntBroker, and so on.

The following are the data types of the message classes:

- Message
- IntBroker
- IBInfo
- IBConnectorInfo

Scope of Message Objects

The message objects can only be instantiated from PeopleCode. These objects can be used anywhere you have PeopleCode, that is, in Component Interface PeopleCode, notification PeopleCode, record field PeopleCode, application engine PeopleCode, and so on.

Message Object Population

After you've declared and instantiated your message object, you want to populate it with data. If your data is coming from the component buffers, populating your message is easy.

A message definition can contain a hierarchy of records. A component buffer contains a hierarchy of records. If you want to copy data from a component buffer rowset to a message, the *structure* of the message and the component must be the same. That is, if you have a record at level two in your message and you want that data, you must have the same level zero and level one records in your message as in your component.

For example, suppose your component had the following structure (that is, that PO_INFO and PO_LINE are at the same level, and PO_DETAIL is the child of PO_INFO):

```
PO_HEADER
   PO_LINE
   PO_INFO
      PO_DETAIL
```

To include the information in the PO_DETAIL record, you must have *at least* the following record structure in your message:

```
PO_HEADER
   PO_INFO
      PO_DETAIL
```

Any records that are in the page that aren't in the message (and vice-versa) are ignored.

After you get your message object, you can create a rowset from it. This rowset has the same structure as the message. If the message is empty, the rowset has no data. If the message has data, the rowset is automatically populated.

The following example is the simplest way of populating a message with data. This assumes that the structure of the message is the same as the structure of the page.

```
/* this gets all the data in the Component buffer */

&RS = GetLevel0();

/* this instantiates a message object */

&MSG = CreateMessage(OPERATION.MY_MESSAGE);

/* creates a rowset with the same structure as the message */

&MSG_RS = &MSG.GetRowset();

/* this copies all the data from the page to the message */

&RS.CopyTo(&MSG_RS);

/* this publishes the message */

%IntBroker.Publish(&MSG_RS);
```

A message rowset is the same as a Component buffer rowset, or any other rowset. It is composed of rows, records, and fields. Suppose you didn't want to get all the data from the Component buffer, but instead wanted to populate just a particular record in your message.

To access a record in a message rowset is the same as accessing a record in a component buffer rowset. You must instantiate the rowset, then specify the row before you can access the record.

The following selects values into a record, then uses the record method CopyFieldTo to copy from the Component buffer record to the message record.

```
Local SQL &LN_SQL;
Local Message &MSG;
Local Rowset &HDR_RS, &LN_RS;
Local Record &LN_REC, &ln_rec_msg;

&MSG = CreateMessage(OPERATION.STOCK_REQUEST);
&HDR_RS = &MSG.GetRowset();
&LN_REC = CreateRecord(Record.DEMAND_INF_INV);

&LN_SQL = CreateSQL("Select * from PS_DEMAND_INF_INV where  BUSINESS_UNIT= :1 and⇒
 ORDER_NO = :2", &BUSINESS_UNIT, &ORDER_NO);

&J = 1;
While &LN_SQL.Fetch(&LN_REC)

  /* copy data into the Level 1 of &MSG object */
  &LN_RS = &HDR_RS(&I).GetRowset(1);

  If &J > 1 Then
    &LN_RS.InsertRow(&J - 1);
  End-If;

  &ln_rec_msg = &LN_RS.GetRow(&J).GetRecord(Record.DEMAND_INF_INV);
  &LN_REC.CopyFieldsTo(&ln_rec_msg);
  &J = &J + 1;
End-While;
```

This section also discusses items to keep in mind when:

- Populating a rowset from a message.
- Publishing and subscribing to partial records.
- Subscribing to character fields.

Considerations When Populating a Rowset From a Message

Suppose your message had two rowsets, one at level zero, a second at level one. In the message, only the level zero rowset contains any data. When you use GetRowset to create a rowset for the entire message, the rowset at level one will contain an empty row, even if there isn't any data in it. (This is standard behavior for *all* rowsets.) However, you can use the IsChanged property on the record object to determine the status of the data.

The following notification PeopleCode example traverse the rowset. (Notice the use of ChildCount, ActiveRowCount, and IsChanged properties).

'. . .' is where application specific code would go.

```
&MSG_ROWSET = &MSG.GetRowset();
For &A0 = 1 To &MSG_ROWSET.ActiveRowCount

/**************************/
/* Process Level 1 Records */
/*------------------------*/

  If &MSG_ROWSET(&A0).ChildCount > 0 Then
  For &B1 = 1 To &MSG_ROWSET(&A0).ChildCount

    &LEVEL1_ROWSET = &MSG_ROWSET(&A0).GetRowset(&B1);
      For &A1 = 1 To &LEVEL1_ROWSET.ActiveRowCount
        If &LEVEL1_ROWSET(&A1).GetRecord(1).IsChanged Then
          . . .

/**************************/
/* Process Level 2 Records */
/*------------------------*/

          If &LEVEL1_ROWSET(&A1).ChildCount > 0 Then
          For &B2 = 1 To &LEVEL1_ROWSET(&A1).ChildCount
            &LEVEL2_ROWSET = &LEVEL1_ROWSET(&A1).GetRowset(&B2);
              For &A2 = 1 To &LEVEL2_ROWSET.ActiveRowCount
                If &LEVEL2_ROWSET(&A2).GetRecord(1).IsChanged Then
                .  .  .

/**************************/
/* Process Level 3 Records */
/*------------------------*/

                If &LEVEL2_ROWSET(&A2).ChildCount > 0 Then
                  For &B3 = 1 To &LEVEL1_ROWSET(&A2).ChildCount
                    &LEVEL3_ROWSET = &LEVEL2_ROWSET(&A2).GetRowset(&B3);
                    For &A3 = 1 To &LEVEL3_ROWSET.ActiveRowCount
                      If &LEVEL3_ROWSET(&A3).GetRecord(1).IsChanged Then
                      .  .  .

                      End-If; /* A3 - IsChanged */
                    End-For; /* A3 - Loop */
                  End-For; /* B3 - Loop */
                End-If; /* A2 - ChildCount > 0 */

/*--------------------------------*/
/* End of Process Level 3 Records */
/**********************************/

              End-If; /* A2 - IsChanged */
            End-For; /* A2 - Loop */
          End-For; /* B2 - Loop */
        End-If; /* A1 - ChildCount > 0 */

/*--------------------------------*/
/* End of Process Level 2 Records */
/**********************************/

      End-If; /* A1 - IsChanged */
    End-For; /* A1 - Loop */
   End-For; /* B1 - Loop */
  End-If; /* A0 - ChildCount > 0 */

/*--------------------------------*/
/* End of Process Level 1 Records */
/**********************************/
```



```
End-For; /* A0 - Loop */
```

Considerations for Publishing and Subscribing to Partial Records

If you've selected to not publish all the fields in the message, you must be careful when inserting that data into a record.

Deselecting the Include check box in the message definition means that the field is *excluded* from the message definition.

- The field won't be included when generating an XML document (when publishing a message.)
- If the field is present in the XML (that is, it wasn't excluded from the published message) it is ignored by the subscribing system.

When you insert the data from the message into the database, you *must* set the values for the fields that aren't in the message. You can use the SetDefault record class method to do this. You could also use a Component Interface based on the component the message was created from to leverage the component defaults.

See Also

Chapter 34, "Record Class," SetDefault, page 2099

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Creating Component Interface-Based Services"

Considerations When Subscribing to Character Fields

If a message definition has character fields that are defined as uppercase, when the message is subscribed to, character data for those fields is automatically converted to uppercase.

Message Segments

To make processing more efficient, you can divide a large message into pieces using message segments. Generally, you only divide asynchronous messages into segments.

Message nodes can be specified as "segment aware". If a node is not segment aware and you send an asynchronous message that is segmented to it, you received an error message when viewing the error message log in message details on the message monitor. No publication contracts are created. If you send a synchronous message that is segmented to a node that is not segment aware, you receive an error.

There are several methods for creating, updating, and deleting segments. There are also two properties that you need to take into consideration when working with segments.

If you specify the SegmentsByDatabase property as false, you can only have the configured number defined in PSADMIN (Message Segment From DB). If you specify this property as true, you can have as many segments as you need.

The SegmentsByDatabase property also specifies whether the segments are kept in memory or written to the database when they are received. If you specify true, the segments are automatically written to the database. If you specify false, the segments are held in memory. If you specify true, then cancel out of the program processing the segments, the changes are not committed to the database.

The SegmentUnOrder property is only applicable for asynchronous messages. If you specify the SegmentUnOrder property as true, the receiving node processes the segments in parallel.

Note. You should use DeleteSegment and UpdateSegment only when writing to memory, or when SegmentsByDatabase is set to False. These methods do not work when writing to the database, or when SegmentsByDatabase is set to True.

The following is an example of how to use the segment properties and methods to send a segmented message. Note that there are only two CreateNextSegment calls. By default the first segment is automatically created. The first time you use CreateNextSegment, the message is split into two segments. The next time, you add a third segment. You don't need to call CreateNextSegment to access the third segment, it's automatically generated.

```
Local Message &MSG;
Local Rowset &FLIGHT_PROFILE, &RS;
Local boolean &Bo, &Stuff;
Local string &lip;

&nodes = CreateArray("");
&nodes[1] = "QE_YO";
&nodes[2] = "QE_STUFF";

QE_FLIGHTDATA.QE_ACNUMBER.Value = QE_FLIGHTDATA.QE_ACNUMBER + 1;

&FLIGHT_PROFILE = GetLevel0();

&MSG = CreateMessage(OPERATION.QE_FLIGHTPLAN);

/* the next lines copy the rowset into the first segment */

&MSG.CopyRowset(&FLIGHT_PROFILE);
&MSG.CreateNextSegment();

/* This copies the next portion of the rowset into the next segment */

&MSG.CopyRowset(&FLIGHT_PROFILE);
&MSG.CreateNextSegment();

/* This copies the last portion of the rowset into the third segment */

&MSG.CopyRowset(&FLIGHT_PROFILE);

/* This specifies that the message segments can be processed separately */

&MSG.IBInfo.SegmentsUnOrder = True;

%IntBroker.publish(&MSG);
```

The following is an example of receiving a segmented message. This would be found in a notification PeopleCode program:


```
Local Message &MSG;
Local Rowset &rs, &rs1;
Local Record &FLIGHTDATA, &REC;

Local string &acnumber_value, &msi_sensor_value, &ofp_value, &actype_value,⇒
 &callsign_value, &squadron_value, &comm1_value, &comm2_value, &ecm_value;

Local XmlDoc &xmldoc;
Local string &yo;
Local boolean &bo;

&CRLF = Char(13) | Char(10);

&MSG = %IntBroker.GetMessage();

/* It is very important to set the rowset to null for every iteration */

/* Also, you may want to verify if the segment count is
greater than 10, and if it is, set the SegmentsByDatabase property to True */

For &i = 1 To &MSG.SegmentCount
   &rs = Null;
   &MSG.GetSegment(&i);

   &rs = &MSG.GetRowset();
   &REC = &rs(1).QE_FLIGHTDATA;

   &FLIGHTDATA = CreateRecord(Record.QE_FLIGHTDATA);
   &REC.CopyFieldsTo(&FLIGHTDATA);

   /* Parse out Message Data */
   &acnumber_value = &FLIGHTDATA.QE_ACNUMBER.Value;
   &msi_sensor_value = &FLIGHTDATA.QE_MSI_SENSOR.Value;
   &ofp_value = &FLIGHTDATA.QE_OFP.Value;
   &actype_value = &FLIGHTDATA.QE_ACTYPE.Value;
   &callsign_value = &FLIGHTDATA.QE_CALLSIGN.Value;
   &squadron_value = &FLIGHTDATA.QE_SQUADRON.Value;
   &comm1_value = &FLIGHTDATA.QE_COMM1.Value;
   &comm2_value = &FLIGHTDATA.QE_COMM2.Value;
   &ecm_value = &FLIGHTDATA.QE_ECM.Value;

   &outstring = "Send Async FLight test";

   /* Construct Output String */
   &outstring = &outstring | &acnumber_value | &CRLF | &msi_sensor_value | &CRLF |⇒
 &ofp_value | &CRLF | &actype_value | &CRLF | &callsign_value | &CRLF | &squadron_⇒
value | &CRLF | &comm1_value | &CRLF | &comm2_value | &CRLF | &ecm_value;

   /* Log Output String into page record */
   &FLIGHTDATA.GetField(Field.DESCRLONG).Value = &outstring;

   SQLExec("DELETE FROM PS_QE_FLIGHTDATA");
   &FLIGHTDATA.Insert();

End-For;
```

See Also

Chapter 25, "Message Classes," CreateNextSegment, page 1266

Chapter 25, "Message Classes," SegmentsByDatabase, page 1308

Chapter 25, "Message Classes," SegmentsUnOrder, page 1371

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker Administration, "Adding and Configuring Nodes"

Content-Based Routing

With PeopleSoft Integration Broker, you typically define the routing information separately from the message itself. This allows you to apply multiple routings to a message and change the routings independent of the message definition.

With content-based routing, attributes of the message are used to make routing decisions. The attributes are set before the message is published. Then, within your implementation of the IRouter.OnRouteSend method, these attributes can be examined and evaluated—for example, to send the message to a defined list of nodes instead of to all nodes. Because the attributes are separate from the message data, the Message object itself does not have to be parsed and loaded into a rowset and then searched to get the routing information. This separation of routing attributes from the message data provides a performance improvement over having those routing attributes within the message content.

In the following example, the routing attributes are set with calls to the AddAttribute method of the IBInfo class:

```
Local Message &MSG;
Local Rowset &FLIGHT_PROFILE;
Local string &AC_Type, &Pilot;
Local boolean &bRet;

&FLIGHT_PROFILE = GetLevel0();

&MSG = CreateMessage(Operation.QE_FLIGHTPLAN);

&AC_Type = GetLevel0().GetRow(1).GetRecord(Record.QE_FLIGHTDATA).AC_TYPE.Value;
&Pilot = GetLevel0().GetRow(1).GetRecord(Record.QE_FLIGHTDATA).PILOT.Value;

&bRet = &MSG.IBInfo.AddAttribute("ACType", &AC_Type);
&bRet = &MSG.IBInfo.AddAttribute("Pilot", &Pilot);

&MSG.CopyRowset(&FLIGHT_PROFILE);

%IntBroker.Publish(&MSG);
```

Then, in the implementation-specific IRouter.OnRouteSend method, the message attributes are evaluated to make routing decisions. The OnRouteSend method does not load and examine the Message object itself, only the attributes:


```
import PS_PT:Integration:IRouter;

class RoutingHandler implements PS_PT:Integration:IRouter;
   method RoutingHandler();
   property array of any destinationNodes;
   method OnRouteSend(&MSG As Message) Returns integer;
   method GetDestinationList(&AC_Type As string, &Pilot As string) Returns any;

end-class;

/* constructor */
method RoutingHandler
end-method;

method OnRouteSend
   /+ &MSG as Message +/
   /+ Returns Integer +/
   /+ Extends/implements PS_PT:Integration:IRouter.OnRouteSend +/
   /* Variable Declaration */
   Local string &AC_Type;
   Local string &Pilot;
   Local any &aNodeList;
   Local integer &i;

   If &MSG.IBInfo.GetNumberOfAttributes() = 0 Then
      Return (%IntBroker_ROUTE_NONE);
   End-If;

   For &i = 1 To &MSG.IBInfo.GetNumberOfAttributes()

      If &MSG.IBInfo.GetAttributeName(&i) = "ACType" Then
         &AC_Type = &MSG.IBInfo.GetAttributeValue(&i);
      End-If;

      If &MSG.IBInfo.GetAttributeName(&i) = "Pilot" Then
         &Pilot = &MSG.IBInfo.GetAttributeValue(&i);
      End-If;

   End-For;
   /* method evaluates these strings and return NodeList */
   &aNodeList = %This.GetDestinationList(&AC_Type, &Pilot);

   Evaluate &aNodeList
   When "True"
      Return (%IntBroker_ROUTE_ALL);
      Break;
   When "False"
      Return (%IntBroker_ROUTE_NONE);
      Break;
   When-Other
      &destinationNodes = &aNodeList.Clone();
      Break;
   End-Evaluate;

end-method;

method GetDestinationList
   /+ &AC_Type as String, +/
   /+ &Pilot as String +/
   /+ Returns Any +/

   /* determine node list based on input data */

   Return Null;
```

```
end-method;
```

See Also

Chapter 25, "Message Classes," AddAttribute, page 1339

Chapter 25, "Message Classes," GetAttributeName, page 1351

Chapter 25, "Message Classes," GetAttributeValue, page 1352

Chapter 25, "Message Classes," GetNumberOfAttributes, page 1354

Error Handling

In your notification PeopleCode, you may want to validate the information received in the message. If you find that the data isn't valid, use Exit(1) to end your PeopleCode program. This sets the status of the message to ERROR, logs any error messages to the Application Message Queues, and automatically rolls back any database changes you may have made.

There are many ways to validate the data and capture the error messages:

- Use ExecuteEdits on the message object
- Use ExecuteEdits on the record object
- Use a Component Interface

Write your own validation PeopleCode in the notification program:

The following example validates the Business Unit of a message against an array of valid BU's:

```
For &I = 1 To &ROWSET.RowCount;

  &POSITION = &BUArray.Find(&ROWSET(&I).GetRecord(1).BUSINESS_UNIT.Value);
  If &POSITION = 0 Then
    &Status = "ERROR: BU not Found or not Active";
    &ROWSET(&I).BCT_ADJS_MSG_VW.BUSINESS_UNIT.IsEditError = True;
    &ROWSET(&I).BCT_ADJS_MSG_VW.BUSINESS_UNIT.MessageSetNumber = 11100;
    &ROWSET(&I).BCT_ADJS_MSG_VW.BUSINESS_UNIT.MessageNumber = 1230;
/* The error message 11100-1230 reads: This Business Unit is */
/* not a valid, open, Inventory Business Unit */

  End-If;

End-For;
```

In the calling PeopleCode, the program calls Exit(1) based on the value of &Status.

Note. All errors for notification PeopleCode get logged to the application message error table, *not* to the PSMessages collection (on the session object.)

See Also

Chapter 25, "Message Classes," ExecuteEdits, page 1268

Chapter 34, "Record Class," ExecuteEdits, page 2086

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Processing Inbound Errors

Message Classes Reference

This reference section documents the following message classes and functions:

- Message class built-in functions.
- Message class.
- IntBroker class.
- IBInfo class.
- IBConnectorInfo collection.

Message Class Built-In Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," AddSystemPauseTimes

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateMessage

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," DeleteSystemPauseTimes

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," PingNode

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReValidateNRXmlDoc

Message Class Methods

In this section, we discuss the Message class methods. The methods are discussed in alphabetical order.

Clone

Syntax

Clone()

Description

The Clone method creates an identical copy of the message, copying the data tree of the message executing the method. The Clone function sets the following properties for the resulting message object, based on the values set for the message definition created in Pure PeopleSoft Internet Architecture:

- Name
- QueueName
- IsOperationActive

Other properties are set when the message is published or subscribed to (TransactionID, PubNodeName, and so on), or are dynamically generated at other times (Size, IsEditError, and so on.)

Clone creates a unique version of the message. If the original message changes, it is not reflected in the cloned object.

Parameters

None.

Returns

A message object.

Example

Clone could be used in a Hub and Spoke messaging environment. The Hub node uses this method during notification processing to publish a copy of the messages it receives from the Spokes.

```
&Msg = %IntBroker.GetMessage();
&Rowset = &Msg.GetRowset();
&Clone = &Msg.Clone();

%IntBroker.Publish(&Clone);
```

The hub's publication routing rules are then used to determine which spokes receive the message.

See Also

Chapter 25, "Message Classes," CopyRowset, page 1260

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateMessage

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

CopyPartRowset

Syntax

`CopyPartRowset``(`*`PartIndex,&Rowset`*`)`

Description

Use the CopyPartRowset to copy the rowset specified by *&Rowset* to the part of the message specified by *PartIndex*.

Note. This method only works with rowset-based container messages.

The primary record of the level zero rowset for both the specified message part and the specified rowset must be the same.

Parameters

Parameter	***Description***
PartIndex	Specify the number of the part you want to copy the rowset data to.
&Rowset	Specify an already instantiated rowset object that contains the data that you want to copy to the message part.

Returns

None.

See Also

Chapter 25, "Message Classes," GetPartRowset, page 1275 and Chapter 25, "Message Classes," GetPartXMLDoc, page 1276

CopyRowset

Syntax

`CopyRowset(source_rowset [, record_list]);`

Where *record_list* is a list of record names in the form:

`RECORD.source_recname1, RECORD.target_recname1`

`[, RECORD.source_recname2, RECORD.target_recname2]. . .`

Description

The CopyRowset method copies data from the source rowset to the like-named fields in the message object executing the method. This is an easy way to populate a message with data from a component.

Note. CopyRowset does not copy effective dated fields.

Note. CopyRowset copies the data, including rows that haven't been modified. If you want to copy only data that has changed in some way, use the CopyRowsetDelta method.

See Chapter 25, "Message Classes," CopyRowsetDelta, page 1261.

When the record names in the message and component do not match exactly, use the optional *record_list* to specify the records to be copied from the component into the message.

When you use the CopyRowset method to copy the contents from the source rowset to the message object, you are creating a *unique* copy of the object. If you change the original rowset, the message object is not changed.

Note. You can execute CopyRowset against a message only once. After a message is populated, any other CopyRowsets are ignored. If you have to populate different levels of a message rowset separately, you can use the CreateRowset method to create a rowset that has the same structure as your message, populate the created rowset, then use CopyRowset to populate your message. You can also use the Rowset CopyTo method to populate a message rowset, then populate the PSCAMA record by hand.

Parameters

Parameter	*Description*
source_rowset	Specifies the name of the rowset to be copied into the message object.
record_list	Specifies specific source and target records to be copied into the message object.

Returns

None.

Example

The following example copies an entire component into a message object.

```
&Msg = %IntBroker.GetMessage();
&Rowset = &Msg.GetRowset()
&Component_Rowset = GetLevel0();
&Msg.CopyRowset(&Component_Rowset);
```

The following example copies a header/line page rowset into a header/line message object, using the *record_list* because the record names don't match:

```
&Msg = %IntBroker.GetMessage();
&Rowset= &Msg.GetRowset();
&Component_Rowset = GetLevel0();
&Msg.CopyRowset(&Component_Rowset, RECORD.PO_HDR_VW, RECORD.PO_HDR, ⇒
RECORD.PO_LINE_VW, RECORD.PO_LINE);
```

See Also

Chapter 25, "Message Classes," CopyRowsetDelta, page 1261

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

Chapter 36, "Rowset Class," page 2127

Chapter 36, "Rowset Class," Fill, page 2135

CopyRowsetDelta

Syntax

CopyRowsetDelta(*source_rowset* [, *record_list*]);

Where *record_list* is a list of record names in the form:

[RECORD.*source_recname1*, RECORD.*target_recname1*

[, RECORD.*source_recname2*, RECORD.*target_recname2*]]. . .

Description

The CopyRowsetDelta method copies rows of data that have changed from the source rowset to the like-named records and like-named fields in the message object executing the method.

Note. CopyRowsetDelta copies all the like-named fields from the changed *row*into the message. It is *not* copying just the changed like-named fields.

When the record names in the message and component do not match exactly, use the optional *record_list* to specify the records to be copied from the component into the message. The specified target records must be records in your message, while the specified source records must be records in a rowset that exists in the data buffer and is populated.

This is an easy way to populate a message when the records in the message match the records in the component that the message is published from.

In addition, the CopyRowsetDelta method sets the AUDIT_ACTN field in the PSCAMA table for every row in the message. The notification process can then use PSCAMA.AUDIT_ACTN to determine how to process every row that was published.

The set values match those used in audit trail processing, that is:

- A - Row inserted.
- D - Row deleted.
- C - Row changed (updated), but no key fields changed. The system copies the new value to the Message rowset.
- K - Row changed (updated), and at least one key field changed. The system copies the old value to the Message rowset and the new value (see "N").
- N - Row changed (updated), and at least one key field changed. The system copies the old value to the Message rowset and the new value (see "K").
- "" − blank value means that nothing on that row has changed. This is the default value, and is the value used to tag the parent rows of children that have changed.

Note. If a child row is inserted (or changed or deleted) CopyRowsetDelta also copies the parent row (and the parent row of that row, and so on, up to the top row) so the subscriber has a full picture of the transaction. A blank value is set in the AUDIT_ACTN field for these rows to let the subscriber know they don't have to take action; the parent rows are there for reference only.

The Audit_Action values "A", "C", "D" are set when a record is added, changed, or deleted, respectively. In some cases such as effective-dated records, the user may change a key field value, such as Effective Date. In response to such an user action, two records are created, one with an Audit_Action value of "N", and the other with Audit_Action value "K". The "N" record has all the new values, while the "K" record retains the old values.

When you use the CopyRowsetDelta method to copy the contents from the source rowset to the message object, you are creating a *unique* copy of the object. If you change the original rowset, the message object is not be changed.

Note. You can execute CopyRowsetDelta against a message only once. After a message is populated, any other CopyRowsetDeltas are ignored. If you have to populate different levels of a message rowset separately, you can use the CreateRowset method to create a rowset that has the same structure as your message, populate the created rowset, then use CopyRowsetDelta to populate your message. You can also use the Rowset CopyTo method to populate a message rowset, then populate the PSCAMA record by hand.

Parameters

Parameter	*Description*
source_rowset	Specifies the name of the rowset to be copied into the message object.
record_list	Specifies source and target records to be copied into the message object. The target records must be records in your message, while the source records must be records in a rowset that exists in the data buffer and is populated.

Returns

None.

Example

The following example copies all the changed rows of data from a component into a message object.

```
&Component_Rowset = GetLevel0();
```

```
&Msg.CopyRowsetDelta(&Component_Rowset);
```

See Also

Chapter 25, "Message Classes," CopyRowset, page 1260

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Understanding Supported Message Structures," PSCAMA

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

Chapter 36, "Rowset Class," page 2127

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

CopyRowsetDeltaOriginal

Syntax

CopyRowsetDeltaOriginal*(source_rowset* [, *record_list*]);

Where *record_list* is a list of record names in the form:

```
[RECORD.source_recname1, RECORD.target_recname1
```

```
[, RECORD.source_recname2, RECORD.target_recname2]]. . .
```


Description

The CopyRowsetDeltaOriginal method copies rows of data that have changed from the source rowset to the like-named records and like-named fields in the message. It also copies the original value of the changed rows.

Note. CopyRowsetDeltaOriginal copies all the like-named fields from the changed and original *rows* into the message. It is not copying just the changed like-named fields.

When the record names in the message and component do not match exactly, use the optional *record_list* to specify the records to be copied from the component into the message. The specified target records must be records in your message, while the specified source records must be records in a rowset that exists in the data buffer and is populated.

The CopyRowsetDeltaOriginal method sets the AUDIT_ACTN field in the PSCAMA table for every row in the message. The notification process can then use PSCAMA.AUDIT_ACTN to determine how to process every row that was published.

The set values match those used in audit trail processing, that is:

- A - Row inserted.
- D - Row deleted.
- C - Row changed (updated), but no key fields changed. The system copies the new value to the Message rowset and the original value (see "O" below).
- O - Prior value of changed row. The system copies the new value to the Message rowset and the original value (see "C").
- K - Row changed (updated), and at least one key field changed. The system copies the original value to the Message rowset and the new value (see "N").
- N - Row changed (updated), and at least one key field changed. The system copies the original value to the Message rowset and the new value (see "K").
- "" − blank value means that nothing on that row has changed. This is the default value, and is the value used to tag the parent rows of children that have changed.

Note. If a child row is inserted (or changed or deleted) CopyRowsetDeltaOriginal also copies the parent row (and the parent row of that row, and so on, up to the top row) so the subscriber has a full picture of the transaction. A blank value is set in the AUDIT_ACTN field for these rows to let the subscriber know they don't have to take action; the parent rows are there for reference only.

The Audit_Action values "A", "C", "D" are set when a record is added, changed or deleted, respectively. When a row is changed, two records are created, one with an Audit_Action of "C", the other with Audit_Action value of "O". The "C" record has all the new values, while the "O" record retains the original values. In some cases such as effective-dated records, a user may change a key field value, such as Effective Date. In response to such an user action, two records are created, one with an Audit_Action value of "N", and the other with Audit_Action value "K". The "N" record has all the new values, while the "K" record retains the original values. The "N"/"K" combination is also created if both a key change and a non-key change is made on the same row of data.

The following table details the output from CopyRowsetDeltaOriginal:

User Action	*CopyRowsetDeltaOriginal Output*	*Comments*
Change a key.	"N" and "K" rows created.	The "K" row contains the original Key value, the "N" row, the new.
Change a Key + change a Non-Key.	"N" and "K" rows created only (that is: no "C"/"O" rows in that case).	The Key and Non-Key Original Values are both contained in the "K" row.
Change a Non-Key.	"C" and "O" rows created.	Original Value stored in "O" row.
Add non-effective dated row.	"A" row created.	No additional rows created.
Add effective-dated row and only 1 original effective-dated row exists.	"A" and "O" rows created.	"O" row contains the original effective dated row.
Add effective-dated row and more than 1 original effected dated rows exist.	"A" and "O" rows created.	"O" row contains data from row that cursor was on when the new row was added. For example: If a rowset contains two rows (01/01/1980 and 02/02/2000), the current row is the 01/01/1980 row and the user adds a new row with today's date. The original values contain the data from the 01/01/1980. Likewise, if a rowset contains two rows (01/01/1980 and 02/02/2000), the current row is the 02/02/2000 row and the user adds a new row with today's date. Then the original values contain the data from the 02/02/2000.
Delete a row	"D" row created.	No additional rows created.

When you use the CopyRowsetDeltaOriginal method to copy the contents from the source rowset to the message object, you are creating a unique copy of the object. If you change the original rowset, the message object is not changed.

Note. You can execute CopyRowsetDeltaOriginal against a message only once. After a message is populated, any other CopyRowsetDeltaOriginal PeopleCode statements are ignored. If you have to populate different levels of a message rowset separately, you can use the CreateRowset method to create a rowset that has the same structure as your message, populate the created rowset, then use CopyRowsetDeltaOriginal to populate your message. You can also use the Rowset CopyTo method to populate a message rowset, then populate the PSCAMA record manually.

Parameters

Parameter	*Description*
source_rowset	Specifies the name of the rowset to be copied into the message object.
record_list	Specifies source and target records to be copied into the message object. The target records must be records in your message, while the source records must be records in a rowset that exists in the data buffer and is populated.

Returns

None.

Example

```
Function Update_Dir(&MSGName As string)

  &MSGName = "OPERATION." | &MSGName;
  &MSG = CreateMessage(@(&MSGName));

  &PnlRowset = GetLevel0();
  &MSG.CopyRowsetDeltaOriginal(&PnlRowset);
  %IntBroker.Publish(&MSG);

End-Function;
```

See Also

Chapter 25, "Message Classes," CopyRowset, page 1260 and Chapter 25, "Message Classes," CopyRowsetDelta, page 1261

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Understanding Supported Message Structures," PSCAMA

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

Chapter 36, "Rowset Class," page 2127

CreateNextSegment

Syntax

`CreateNextSegment``()`

Description

Use the CreateNextSegment method to divide a message into segments. You generally only divide asynchronous messages that contain a large amount of data. This method is used only when creating a message for publication, not after a message has been published.

Parameters

None.

Returns

None.

Example

See Chapter 25, "Message Classes," Message Segments, page 1251.

See Also

Chapter 25, "Message Classes," CreateNextSegment, page 1266; Chapter 25, "Message Classes," DeleteSegment, page 1267; Chapter 25, "Message Classes," SegmentRestart, page 1285; Chapter 25, "Message Classes," UpdateSegment, page 1294; Chapter 25, "Message Classes," CurrentSegment, page 1297 and Chapter 25, "Message Classes," SegmentCount, page 1308

DeleteSegment

Syntax

`DeleteSegment`*`(SegmentNumber)`*

Description

Use the DeleteSegment method to delete the specified segment. This method is used only when creating a message for publication, not after a message has been published.

Note. You should use DeleteSegment and UpdateSegment only when writing to memory, or when SegmentsByDatabase is set to False. These methods do not work when writing to the database, or when SegmentsByDatabase is set to True.

Parameters

Parameter	*Description*
SegmentNumber	Specify a number indicating which segment you want to delete from the unpublished message.

Returns

None.

See Also

Chapter 25, "Message Classes," CreateNextSegment, page 1266; Chapter 25, "Message Classes," GetSegment, page 1280; Chapter 25, "Message Classes," UpdateSegment, page 1294; Chapter 25, "Message Classes," CurrentSegment, page 1297 and Chapter 25, "Message Classes," SegmentCount, page 1308

ExecuteEdits

Syntax

`ExecuteEdits``([`*`editlevels`*`]);`

where *editlevels* is a list of values in the form:

`editlevel1`` [+ `*`editlevel2`*`] . . .]);`

and where *editleveln* is one of the following system constants:

%Edit_DateRange

%Edit_OneZero

%Edit_PromptTable

%Edit_Required

%Edit_TranslateTable

%Edit_YesNo

Description

The ExecuteEdits method executes the standard system edits on every field for every record in the message executing the method. All edits are based on the record edits defined for the record that the message is based on. The types of edits performed depends on the *editlevel.* If no *editlevel* is specified, all system edits defined for the record definition are executed, that is:

- Reasonable Date Range (Is the date contained within plus or minus 30 days from the current date?)
- 1/0 (Do all 1/0 fields only contain 1 or 0?)
- Prompt Table (Is field data contained in the specified prompt table?)
- Required Field (Do all required fields contain data? For numeric or signed fields, it checks that they do not contain NULL or 0 values.)
- Translate Table (Is field data contained in the translate table?)
- Yes/No (Do all yes/no fields only contain only yes or no data?)

Note. ExecuteEdits does not perform any validation on DateTime fields.

If any of the edits fail, the status of the property IsEditError is set to True for the Message Object executing the method, and any Rowset, Row, Record, or Field Objects that could be instantiated from the same data as the Message Object. In addition, the Field class properties MessageNumber and MessageSetNumber are set to the number of the returned message and message set number of the returned message, for the field generating the error.

If you want to check the field values only for a particular record, you can use the ExecuteEdits method with a record object.

In addition, if you want to dynamically set the prompt tables used for a field (with the %EditTable function) you must use the SetEditTable method.

Considerations for ExecuteEdits and SetEditTable

If an effective date is a key on the prompt table, and the record being edited doesn't contain an effective-dated field, the current datetime is used as the key value.

If a setID is a key on the prompt table, and the record being edited doesn't contain a setID field, the system looks for the setID on the other records in the current row first, then search parent rows.

Considerations for ExecuteEdits and Numeric Fields

A zero (0) might or might not be a valid value for a numeric field. ExecuteEdits processes numeric fields in different ways, depending on whether the field is required:

- If the numeric field is required: 0 is considered invalid.
- If the numeric field is not required: 0 is considered valid.

Parameters

Parameter	*Description*
editlevel	Specifies the standard system edits to be performed against every field on every record. If *editlevel* isn't specified, all system edits are performed. *editlevel* can be any of the following system constants: • %Edit_DateRange • %Edit_OneZero • %Edit_PromptTable • %Edit_Required • %Edit_TranslateTable • %Edit_YesNo

Returns

None.

Example

The following is an example of a call to execute Required Field and Prompt Table edits:

```
&MSG.ExecuteEdits(%Edit_Required + %Edit_PromptTable);
```

The following is an example showing how ExecuteEdits() could be used:

```
&MSG.ExecuteEdits();
If &MSG.IsEditError Then
  Exit(1);
End-If;
```

See Also

Chapter 25, "Message Classes," SetEditTable, page 1286

Chapter 18, "Field Class," EditError, page 951

Chapter 18, "Field Class," MessageNumber, page 962

Chapter 18, "Field Class," MessageSetNumber, page 963

Chapter 34, "Record Class," IsEditError, page 2106

Chapter 34, "Record Class," ExecuteEdits, page 2086

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Processing Inbound Errors

GenXMLPartString

Syntax

`GenXMLPartString`*`(PartIndex)`*

Description

Use the GenXMLPartString to create an XML string based on the message part specified by *PartIndex.*

Note. This method only works with rowset-based container messages.

Parameters

Parameter	*Description*
PartIndex	Specify the number of the part that you want to generate an XML string for.

Returns

An XML string.

See Also

Chapter 25, "Message Classes," GenXMLString, page 1271; Chapter 25, "Message Classes," GetPartRowset, page 1275 and Chapter 25, "Message Classes," LoadXMLPartString, page 1282

GenXMLString

Syntax

`GenXMLString``()`

Description

The GenXMLString method creates an XML string based on the message after it's loaded from the message buffer.

Note. This method does not support nonrowset-based messages.

Parameters

None.

Returns

An XML string.

Example

In the following example, the first thee lines of code creates the message object and copies the data from a Component buffer into the message object. The last line creates an XML string based on that message object.

```
&MSG = CreateMessage(OPERATION.PO_INSERT);
&RS = GetLevel0();
&MSG.CopyRowset(&RS);
&XML_STRING = &MSG.GenXMLString();
```

See Also

Chapter 25, "Message Classes," LoadXMLString, page 1283

GetContentString

Syntax

GetContentString([*SegmentIndex*])

Description

Use the GetContentString to return a string populated with the content from the specified message segment. If you don't specify a segment, the data from the first segment is used.

Parameters

Parameter	***Description***
SegmentIndex	Specify the message segment that you want to access.

Returns

A string populated with the content of the specified segment if successful, null otherwise.

See Also

Chapter 25, "Message Classes," CreateNextSegment, page 1266

GetDocument

Syntax

GetDocument([*segmented_messsage*])

Description

Use these method to get a Document object associated with this message.

Note. If a fault occurs, continue to use GetContentString to read the fault message, and not GetDocument.

Parameters

Parameter	*Description*
segmented_messsage	Specify a Boolean value indicating whether the message is segmented.

Returns

A Document object.

Example

```
Local Message &Msg;
Local Document &Doc;

&Msg = CreateMessage(Operation.QEFLIGHTPLAN_DOC);

&Doc = &Msg.GetDocument( True);

/* populate the document */
...

&Msg.CreateNextSegment();

&Doc = Null;
&Doc = &Msg.GetDocument( True);

/* populate the document */

...
```

See Also

Chapter 25, "Message Classes," GetContentString, page 1272

Chapter 15, "Document Classes," Document Class, page 730

GetPartAliasName

Syntax

GetPartAliasName*(PartIndex)*

Description

Use the GetPartAliasName to access the alias of the specified message part.

Note. This method works only with container messages.

Parameters

Parameter	*Description*
PartIndex	Specify the message part that you want to access the alias.

Returns

A string containing the message part alias.

See Also

Chapter 25, "Message Classes," AliasName, page 1295

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Specifying Record Aliases

GetPartName

Syntax

GetPartName*(PartIndex)*

Description

Use the GetPartName method to return the message name of the message specified by *PartIndex*.

Note. This method works only with container messages.

Parameters

Parameter	***Description***
PartIndex	Specify the number of the message part that you want to get the name for.

Returns

A string.

See Also

Chapter 25, "Message Classes," CopyPartRowset, page 1259 and Chapter 25, "Message Classes," GenXMLPartString, page 1271

GetPartRowset

Syntax

`GetPartRowset`*`(PartIndex)`*

Description

Use the GetPartRowset to instantiate and populate a rowset object based on the specified message part in a container message.

Note. This method only works with rowset-based container messages.

Parameters

Parameter	***Description***
PartIndex	Specify the number of the message part that you want to instantiate and populate a rowset from.

Returns

A reference to a rowset object if successful, Null otherwise.

See Also

Chapter 25, "Message Classes," CopyPartRowset, page 1259 and Chapter 25, "Message Classes," GenXMLPartString, page 1271

GetPartVersion

Syntax

GetPartVersion*(PartIndex)*

Description

Use the GetPartVersion method to return the version number of the message part specified by *PartIndex.*

Note. This method only works with container messages.

Parameters

Parameter	*Description*
PartIndex	Specify the number of the message part that you want to access the version of.

Returns

A string containing the version number.

See Also

Chapter 25, "Message Classes," PartCount, page 1306

GetPartXMLDoc

Syntax

GetPartXMLDoc*(PartIndex)*

Description

Use the GetPartXMLDoc to retrieve the message part data as an XmlDoc object.

Note. This method can only be used for a nonrowset-based message parts.

Parameters

Parameter	***Description***
PartIndex	Specify the message part that you want to access the data for as an XmlDoc object.

Returns

An XmlDoc object populated with the message part data.

See Also

Chapter 25, "Message Classes," GetXmlDoc, page 1280 and Chapter 25, "Message Classes," GetPartRowset, page 1275

GetQueryString

Syntax

```
GetQueryString(Parameter_Name)
```

Description

Use the GetQueryString method to obtain the parameter values of a query string.

Note. This method has been deprecated and remains for backward compatibility only. Use the IBConnectorInfo collection GetQueryStringArg method instead.

See Also

Chapter 25, "Message Classes," GetQueryStringArgName, page 1381

Parameters

Parameter	*Description*
Parameter_Name	Specify the parameter name you want to obtain.

Returns

A string containing the value of the parameter.

Example

```
Local Message &request;
Local Rowset &Rowset;
Local String &emplid;

&request = %IntBroker.GetMessage();
&Rowset = &request.GetRowset();
&emplid = &request.GetQueryString("EMPLID");
```

See Also

Chapter 25, "Message Classes," GetQueryStringArgName, page 1381; Chapter 25, "Message Classes," GetQueryStringArgValue, page 1382 and Chapter 25, "Message Classes," SetQueryString, page 1288

GetRowset

Syntax

GetRowset([*version*])

Description

The GetRowset method returns a standard PeopleCode level zero rowset object for the specified message version. It creates an empty rowset for the specified message version if it doesn't already exist. If no message version is specified, the default message version is used.

When you use the GetRowset method, you are not creating a unique copy of the object. You are making only a copy of the reference. If you change the rowset, the original message is changed.

Parameters

Parameter	*Description*
version	An optional parameter, specifying an existing message version as a string. If *version* isn't specified, the default message version is used.

Returns

A Rowset object if successful.

Example

The following gets all the SET_CNTRL_REC rows related to the row on the page, then updates the field SETID with the value from the page. GetRowset is used to create a rowset based on the message structure, that is, an unpopulated rowset. Because the rowset is unpopulated, you can use the Fill method to populate with data from the page.

```
Local Message &MSG;
Local Rowset &MSG_ROWSET;

If FieldChanged(SETID) Then
  &MSG = CreateMessage(OPERATION.SET_CNTRL_REC);
  &MSG_ROWSET = &MSG.GetRowset();
  &MSG_ROWSET.Fill("where SETCNTRLVALUE =:1 and REC_GROUP_ID =:2", SETCNTRLVALUE,⇒
 REC_GROUP_ID);

  For &I = 1 To &MSG_ROWSET.ActiveRowCount
    &MSG_ROWSET.GetRow(&I).SET_CNTRL_REC.SETID.Value = SETID;
    &MSG_ROWSET.GetRow(&I).PSCAMA.AUDIT_ACTN.Value = "C";
  End-For;

  %IntBroker.Publish(&MSG);

End-If;
```

See Also

Chapter 25, "Message Classes," Clone, page 1258

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateMessage

Chapter 36, "Rowset Class," page 2127

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

GetSegment

Syntax

`GetSegment``(`*`SegmentNumber`*`)`

Description

Use the GetSegment method to return the specified segment. This method doesn't actually return anything, but populates the current message object with the specified segment's data.

Parameters

Parameter	*Description*
SegmentNumber	Specify the number of the segment you want to access.

Returns

None.

See Also

Chapter 25, "Message Classes," CreateNextSegment, page 1266; Chapter 25, "Message Classes," DeleteSegment, page 1267; Chapter 25, "Message Classes," UpdateSegment, page 1294 and Chapter 25, "Message Classes," SegmentCount, page 1308

GetXmlDoc

Syntax

`GetXmlDoc``()`

Description

Use the GetXmlDoc method to retrieve the message data as an XmlDoc object.

Note. This method can only be used for a nonrowset-based message. If you use this method with a rowset-based message an error message is returned.

Parameters

None.

Returns

A XmlDoc object.

See Also

Chapter 25, "Message Classes," SetXmlDoc, page 1290

Chapter 46, "XmlDoc Classes," page 2529

InBoundPublish

Syntax

`InBoundPublish``(`*`PubNodeName`*`)`

Description

Use the InBoundPublish method to send an asynchronous message based on a message rowset to simulate an inbound asyncronous request from an external node.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class InBoundPublish method instead.

Though you are sending a message to yourself, it goes through all the inbound message processing on *PubNodeName.*

See Also

Chapter 25, "Message Classes," InBoundPublish, page 1324

Parameters

Parameter	***Description***
PubNodeName	Specify the name of the node for which you want to test inbound message publishing.

Returns

None.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," InboundPublishXmlDoc

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Applying Filtering, Transformation and Translation"

LoadXMLPartString

Syntax

`LoadXMLPartString``(`*`PartIndex,XmlString`*`)`

Description

Use the LoadXMLPartString method to populate the part of the container message specified by *PartIndex* with the XML string *XmlString.*

Note. This method works only with container messages.

Parameters

Parameter	***Description***
PartIndex	Specify the number of the part of the container message that you want to populate with XML data.
XmlString	Specify the XML string to be loaded into the message part.

Returns

None.

See Also

Chapter 25, "Message Classes," GenXMLPartString, page 1271 and Chapter 25, "Message Classes," GenXMLString, page 1271

LoadXMLString

Syntax

`LoadXMLString``(`*`XMLstring`*`)`

Description

The LoadXMLString method loads the message object executing the method with the XML string *XMLstring.*

Parameters

Parameter	***Description***
XMLString	Specify the XML string to be loaded into the message object.

Returns

None.

Example

```
&MSG = CreateMessage(OPERATION.PO_HEADER);
&MSG.LoadXMLString(&XML_STRING);
```

See Also

Chapter 25, "Message Classes," GenXMLString, page 1271

Publish

Syntax

`Publish``([`*`NodeName`*`] [, `*`Deferred_Mode`*`] )`

Description

The Publish method publishes a message to the application message queue for the default message version.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class Publish method instead.

This method does not publish the message if the IsActive property is False.

This method publishes a message asynchronously, that is, a reply message is not automatically generated. To publish a message synchronously, use the SyncRequest method.

Note. This method supports nonrowset-based messages only if the data is populated using the SetXmlDoc method.

If the Publish method is called and the message isn't populated (either using SetXmlDoc or one of the rowset methods,) an empty message is published.

The Publish method can be called from any PeopleCode event, but is generally called from an Application Engine PeopleCode step or from a component.

When publishing from a component, you should publish messages only from the SavePostChange event, either from record field or component PeopleCode. This way, if there's an error in your publish code, the data isn't committed to the database. Also, since SavePostChange is the last Save event fired, you avoid locking the database while the other save processing is happening.

However, until the message is published, the tables involved with the component are locked and are not saved. To avoid problems with this, specify the *Deferred_Mode* property as true, so the message is published after the table data has been committed to the database.

Generally, if the message is successfully published, the PubID, and PubNodeName properties are set to the publication ID and publishing system node name, respectively. The only exception is when the Publish is performed as part of your notification PeopleCode. The notification process is always executed in deferred mode, due to performance considerations, and so the PubID is not populated.

Note. If you're publishing a message from within an Application Engine program, the message won't actually be published until the calling program performs a Commit.

See Also

Chapter 25, "Message Classes," Publish, page 1326

Parameters

Parameter	***Description***
NodeName	Specify the node to which the message should be published.
Deferred_Mode	Specify whether the message should be published immediately or at the end of the transaction. This parameter takes a Boolean value: False, publish the message immediately, or True, defer the publication. The default value is False.

Returns

None.

Example

The following copies all the data from a page into a message and publishes it.

```
Local Message &MSG;
Local Rowset &ROWSET;

&MSG = CreateMessage(OPERATION.MESSAGE_PO):
&ROWSET = GetLevel0();
&MSG.CopyRowset(&ROWSET);
&MSG.Publish;
```

The following is an example of putting the Incremental Publish PeopleCode on a Record at Level 1 (or 2, 3). If you just do the normal publishing PeopleCode, you get as many messages as there are records at that level (because the code fires for each record).

To make sure the code fires only once, use the following code.

```
/*NAMES.EMPLID.WORKFLOW */
Local Message &MSG;
Local Rowset &RS;

&LEVEL = CurrentLevelNumber();
&CURRENT_ROW = CurrentRowNumber(&LEVEL);
&TOT_ROW = ActiveRowCount(EMPLID);
/* Only Publish the message once for all records on */
/* this level */
If &CURRENT_ROW = &TOT_ROW Then
  &MSG = CreateMessage(OPERATION.NAMES);
  &RS = GetRowset();
  &MSG.CopyRowsetDelta(&RS);
  &MSG.Publish();
End-If;
```

See Also

Chapter 25, "Message Classes," IsActive, page 1299 and Chapter 25, "Message Classes," InBoundPublish, page 1281

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," InboundPublishXmlDoc

SegmentRestart

Syntax

`SegmentRestart`*`(TransactionID,Segment_index,segmentByDB)`*

Description

Use the SegmentRestart method to restart a message segment. This is used only in a Application Engine program, and only if check point logic is used.

Parameters

Parameter	***Description***
Transaction_ID	Specify the transaction ID of the message you want to restart.
Segment_Index	Specify the number of the segment that you want to restart.
SegmentByDB	Specify the segments by database.

Returns

None.

See Also

Chapter 25, "Message Classes," DeleteOrphanedSegments, page 1316

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Understanding Message Segments

SetEditTable

Syntax

```
SetEditTable(%PromptField,RECORD.recordname)
```

Description

The SetEditTable method works with the ExecuteEdits method. It is used to set the value of a field on a record that has its prompt table defined as *%PromptField* value. *%PromptField* values are used to dynamically change the prompt record for a field.

There are several steps to setting up a field so that you can dynamically change its prompt table.

To set up a field with a dynamic prompt table:

1. Define a field in the DERIVED record called *fieldname.*

2. In the record definition for the field that you want to have a dynamic prompt table, define the prompt table for the field as *%PromptField,* where *PromptField* is the name of the field that you created in the DERIVED record.

3. Use SetEditTable to dynamically set the prompt table.

%PromptField is the name of the field on the DERIVED work record. **RECORD.***recordname* is the name of the record to be used as the prompt table.

```
&MSG.SetEditTable("%EDITTABLE1", Record.DEPARTMENT);
&MSG.SetEditTable("%EDITTABLE2", Record.JOB_ID);
&MSG.SetEditTable("%EDITTABLE3", Record.EMPL_DATA);
&MSG.ExecuteEdits();
```

Every field on a record that has the prompt table field %EDITTABLE1 will have the same prompt table, such as, DEPARTMENT.

Parameters

Parameter	***Description***
%PromptField	Specifies the name of the field in the DERIVED record that has been defined as the prompt table for a field in a record definition.
RECORD.*recordname*	Specifies the name of the record definition that contains the correct standard system edits to be performed for that field in the message.

Returns

None.

See Also

Chapter 25, "Message Classes," ExecuteEdits, page 1268

Chapter 18, "Field Class," EditError, page 951

Chapter 18, "Field Class," MessageNumber, page 962

Chapter 18, "Field Class," MessageSetNumber, page 963

Chapter 34, "Record Class," IsEditError, page 2106

Chapter 34, "Record Class," SetEditTable, page 2100

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Record Definitions," Creating New Records

SetQueryString

Syntax

`SetQueryString(``Parameter_Name,Value)`

Description

Use the SetQueryString method to add a parameter value to the query string.

Note. This method has been deprecated and remains for backward compatibility only. Use the IBConnectorInfo class AddQueryStringArg method instead.

See Also

Chapter 25, "Message Classes," AddQueryStringArg, page 1375

Parameters

Parameter	*Description*
Parameter_Name	Specify the name of the parameter that you want to add to the query string, as a string.
Value	Specify the value of the parameter you are adding to the query string.

Returns

None.

Example

```
Local Message &request;
&request = CreateMessage(OPERATION.QE_SALES_ORDER);
. . .

. . .

&request.SetQueryString("BUYER", "12345");
```

See Also

Chapter 25, "Message Classes," AddQueryStringArg, page 1375 and Chapter 25, "Message Classes," GetQueryString, page 1277

SetStatus

Syntax

`SetStatus`*`(Status)`*

Description

Use the SetStatus method to set the status of a publication or subscription contract, much the same way you do in the message monitor.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class SetStatus method instead.

You may want to use this method after an end-user has finished fixing any errors in the message data.

You can set the status of a contract to one of the following statuses:

- Cancel
- Done
- Error
- New

The method is available only when the message instance has one of the following statuses:

- Cancel
- Done
- Error
- New

See Also

Chapter 25, "Message Classes," SetStatus, page 1331

Parameters

Parameter	*Description*
Status	Specify the status that you want to set the message to. Value are: • %Message_Error • %Message_New • %Message_Done • %Message_Started • %Message_Working • %Message_Retry • %Message_Timeout • %Message_Edited • %Message_Canceled

Returns

None.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

SetXmlDoc

Syntax

SetXmlDoc*(&XmlDoc)*

Description

Use the SetXmlDoc method to load a nonrowset-based message with data from an XmlDoc object.

Note. This method can only be used for nonrowset-based message. If you use this method with a rowset-based message an error message is returned.

Considerations Using SetXmlDoc

In order to wrap non-XML data in cdata wraps as part of a message that would be automatically removed when posted to the gateway, the following cdata wraps are supported:

- <?xml version="1.0"?>
  <data PsNonXml="Yes"><![CDATA..
- <?xml version="1.0"?>
  <data psnonxml="Yes"><![CDATA..
- <?xml version="1.0"?><add_some_root_tag>
  <data PsNonXml="Yes"><![CDATA...
- <?xml version="1.0"?><add_some_root_tag>
  <data psnonxml="Yes"><![CDATA...

Parameters

Parameter	*Description*
&XmlDoc	Specify an already instantiated, populated XmlDoc object.

Returns

None.

See Also

Chapter 25, "Message Classes," GetXmlDoc, page 1280

Chapter 46, "XmlDoc Classes," page 2529

SyncRequest

Syntax

SyncRequest([*NodeName*])

Description

Use the SyncRequest method to send a synchronous message for the default message version.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class SyncRequest method instead.

Use the IsActive property to determine if a message is active or not before using this method.

This method sends a single message synchronously, that is a reply message is returned as a result of this method.

If you want to publish a message asynchronously, use the Publish method.

If you want to publish more than one message at a time synchronously, use the IntBroker SyncRequest method.

Note. This method supports nonrowset-based messages only if the SetXmlDoc method is used to populate the message prior to using the SyncRequest method.

The SyncRequest method can be called from any PeopleCode event, but is generally called from an Application Engine PeopleCode step or from a component.

When sending a synchronous request from a component, you should send messages only from the SavePostChange event, either from record field or component PeopleCode. This way, if there's an error in your SyncRequest code, the data isn't committed to the database. Also, since SavePostChange is the last Save event fired, you avoid locking the database while the other save processing is happening.

If the message is successfully sent, a response message is returned. In addition, the GUID property is updated with a unique identifier.

See Also

Chapter 25, "Message Classes," SyncRequest, page 1332

Parameters

Parameter	***Description***
NodeName	Specify the name of the node that you want to send this message to.

Returns

A response message.

Example

```
Local Message &request, &response;
Local Rowset &sales_order;

&sales_order = GetLevel0();
&request = CreateMessage(OPERATION.QE_SALES_ORDER);
&request.CopyRowsetDelta(&sales_order);
&response = &request.SyncRequest();
If (&response.ResponseStatus = 0) Then
   /* do processing */
End-If;
```


See Also

Chapter 25, "Message Classes," Publish, page 1283

Chapter 25, "Message Classes," SyncRequest, page 1332

Update

Syntax

Update([*versionlist*])

where *versionlist* is an arbitrary list of message versions in the following format:

version1 [, *version2*] . . .

Description

The Update method updates a message in the message queue with the specified message versions.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class Update method instead.

If *versionlist* isn't specified, the default message version is used. This method is commonly used in the OnRouteSend and OnRouteReceive PeopleCode events.

Note. This method does not support nonrowset-based messages. The Update method can't be called from notification PeopleCode.

See Also

Chapter 25, "Message Classes," Update, page 1334

Parameters

Parameter	*Description*
versionlist	Specify an optional comma-separated list of message versions. If *versionlist* isn't specified, the default message version is used.

Returns

None.

See Also

Chapter 25, "Message Classes," GetRowset, page 1278

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

UpdateSegment

Syntax

`UpdateSegment``()`

Description

Use the UpdateSegment method to update the current segment with data from the message. This method is used only when creating a message for publication, not after a message has been published.

Note. You should use DeleteSegment and UpdateSegment only when writing to memory, or when SegmentsByDatabase is set to False. These methods do not work when writing to the database, or when SegmentsByDatabase is set to True.

Parameters

None.

Returns

None.

See Also

Chapter 25, "Message Classes," CreateNextSegment, page 1266; Chapter 25, "Message Classes," DeleteSegment, page 1267; Chapter 25, "Message Classes," CurrentSegment, page 1297 and Chapter 25, "Message Classes," SegmentCount, page 1308

Message Class Properties

In this section, we discuss the Message class properties. The properties are discussed in alphabetical order.

ActionName

Description

This property returns the name of the action associated with the message, as a string.

This property is read-only.

AliasName

Description

This property returns the name of the alias associated with the message part executing the property.

This property is read-only.

See Also

Chapter 25, "Message Classes," GetPartAliasName, page 1274

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Specifying Record Aliases

ChannelName

Description

This property references the name of the channel associated with the message definition.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class QueueName property instead.

See Chapter 25, "Message Classes," QueueName, page 1307.

This property is set when the message is created. The message instance keys are a subset of the publication and subscription contract keys. This property is one of the message instance keys: the others are PubID and PubNodeName for asynchronous messages, GUID and PubNodeName for synchronous messages.

This property is valid for both synchronous and asynchronous messages.

This property is read-only.

Example

```
&CHANNEL = &MSG.ChannelName
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

Cookies

Description

This property returns or sets the cookies associated with a message.

Note. This property has been deprecated and remains for backward compatibility only. Use the IBInfo class Cookies property instead.

See Chapter 25, "Message Classes," Cookies, page 1383.

You can accept a synchronous response message containing cookies, save those cookies in a global variable, and later return them to the target node in an outbound synchronous or asynchronous request message.

You can access this property only in an inbound synchronous response message or an outbound request message.

This property is read-write.

Example

The following is an example of receiving cookies:

```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;

&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* send the synchronous request; the return value is */
/* the response message object */
&SalesResponse = &SalesRequest.SyncRequest();

/* Retrieve cookies from the response message */
&SalesCookies = &SalesResponse.Cookies;
```

The following is an example of returning cookies:

```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;

&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* Insert the cookies in the request message */
&SalesRequest.Cookies = &SalesCookies;

/* Send the asynchronous request */
&SalesRequest.Publish();
```

CurrentSegment

Description

This property returns a number, indicating which segment is the current segment.

This property is read-only.

DefaultMessageVersion

Description

This property returns the default version of the message as a string.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class OperationVersion property instead.

See Chapter 25, "Message Classes," OperationVersion, page 1306.

This is the message version from the pub/sub contract, not the default message version of the message defined in the current system.

This property is valid for both synchronous and asynchronous messages.

This property is read-only.

Example

```
Local Message &MSG;
Local String &VER;

&MSG = %IntBroker.GetMessage();
&VER = &MSG.DefaultMessageVersion;
```


DoNotPubToNodeName

Description

Use this property to set the node name of a node that you do not want to publish this message to. For example, a single node may publish and subscribe to the same message. You can use this property to prevent the system from subscribing to the same message it publishes. This can help prevent circular processing where the original publisher eventually receives the same message.

This property is only valid for asynchronous messages.

This property is read-write.

Example

```
&MSG.DoNotPubToNodeName = &MSG.PubNodeName;
&MSG.Publish();
```

GUID

Description

This property returns a string representing a global unique identifier generated when the message was sent.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class TransactionId property instead.

See Chapter 25, "Message Classes," TransactionId, page 1312.

This property returns the unique identifier used with the message.

This property is valid only for synchronous messages.

This property is read-only.

Example

```
Local Message &request;
Local String &guid;

&request = %IntBroker.GetMessage();

&guid = &request.GUID;
```


IBException

Description

Use the IBException property to return a reference to an exception object. The object is populated with information about the integration broker error.

This property is read-only.

See Also

Chapter 16, "Exception Class," page 761

IBInfo

Description

Use this property to return a reference to an IBInfo object.

This property is read-only.

IsActive

Description

Indicates whether the message definition has been set to inactive.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class IsOperationActive property instead.

See Chapter 25, "Message Classes," IsOperationActive, page 1303.

This property is True if the message definition is active, False if it's been inactivated. If you have a lot of PeopleCode associated with publishing a message, you might use this property to check if the message is active before you publish it.

This property is valid for both asynchronous and synchronous messages.

This property is read-only.

Example

```
&MSG = CreateMessage(OPERATION.MY_MESSAGE)
If &MSG.IsActive Then
  /* do PeopleCode processing */
  &MSG.Publish();
Else
  /* do other processing */
End-if;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," IsMessageActive

IsBulkLoadTruncation

Description

Use this property to return a Boolean value indicating whether the truncation option is selected on the bulk loader handler.

This property is read-only.

IsDelta

Description

This property indicates if any fields in a message populated from page data have been changed since they were first loaded into the component buffer.

This is generally used in the following scenario:

You want to publish a message only if the end-user has changed a value on a page. The problem is that if the end-user makes a change to a field in the level three rowset, the IsChanged property for the top level row remains False. If the rowset is large, and you don't want to iterate through it to find a single IsChanged, you can use this property to quickly determine if something has changed at a lower level and the message should be published.

Note. This property should be used only when the message and the component do not have the same structure (that is, the same records at the same levels). If the message and the component have the same structure use the CopyRowsetDelta method instead.

This property takes a Boolean value: True if there are changed fields in the message, False otherwise.

This property is valid for both asynchronous and synchronous messages.

This property is read-only.

Example

```
&Msg = CreateMessage(OPERATION.MY_MESSAGE);
&Msg_Rowset = &Msg.GetRowset(); /* get first level in Msg RS */
&Msg_Detail = &Msg_Rowset(1).GetRowset(); /* get detail in Msg */
&Page_Rowset = GetRowset(); /* get page */
For &I = &Page_Rowset.Rowcount
  &Page_Detail = &Page_Rowset.GetRow(&I).GetRowset();
  &Msg_Detail.CopyRowset(&Page_Detail);
End-For;
/* Find if any data changed and should publish message */
If &Msg.IsDelta Then
   &Msg.Publish();
Else
   /* don't publish message and do other processing */
End-If;
```

See Also

Chapter 25, "Message Classes," CopyRowsetDelta, page 1261

IsEditError

Description

This property is True if an error has been found on any field in any record of the current message after executing the ExecuteEdits method on either a message object or a record object. This property can be used with the Field Class properties MessageSetNumber and MessageNumber to find the error message set number and error message number.

You can use this property in notification PeopleCode with the Exit function with a value of 1. This automatically rolls back any changes that were made to the database, saves the message errors to the message queue, and marks the message status as ERROR.

This property is valid for both asynchronous and synchronous messages.

This property is read-only.

Example

```
&MSG = %IntBroker.GetMessage();
&Rowset = &MSG.GetRowset();
&MSG.ExecuteEdits();
If &MSG.IsEditError Then
  Exit(1);
End-If;
```


See Also

Chapter 25, "Message Classes," ExecuteEdits, page 1268

Chapter 18, "Field Class," MessageNumber, page 962

Chapter 18, "Field Class," MessageSetNumber, page 963

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Exit

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Processing Inbound Errors

IsEmpty

Description

This property indicates whether the transaction occurred without error, but returned an empty XML document. An empty response object contains only the data tags:

```
<data></data>
```

This property returns a Boolean value: True, the response message is empty, False, the response message is not empty.

This property is valid only for the returned response message from a synchronous request message.

This property is read-only.

IsLocal

Description

This property is True if the message object that was just instantiated was published locally. This property takes a Boolean value.

This property is valid for both asynchronous and synchronous messages.

This property is read-only.

Example

```
&MSG = %IntBroker.GetMessage();
If &MSG.IsLocal Then /*  LOCAL NODE */
  /* do processing specific to local node */
Else
  /* do processing specific to remote nodes */
End-If;
```


IsOperationActive

Description

Use the IsOperationActive property to determine if an operation is active or not.

This property returns a Boolean value: true if the operation is still active, false otherwise.

This property is read-only.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

IsParts

Description

Use the IsParts property to determine if the message is a container message, that is, composed of parts.

This property only determines if a message is a container message. If you'd like to know if the message is a rowset-based message with parts, use the IsPartsStructured property instead.

This property returns a Boolean value: true if the message executing the property is a container message, false otherwise.

This property is read-only.

See Also

Chapter 25, "Message Classes," IsPartsStructured, page 1303

IsPartsStructured

Description

Use the IsPartsStructured property to determine if the message is a container message, that is, composed of parts, as well as a rowset-based message.

This property determines if a message is a container message as well as rowset-based. If you only want to know if the message is a container message, use the IsParts property instead.

This property returns a Boolean value: true if the message executing the property is a container message that is rowset-based, false otherwise.

This property is read-only.

See Also

Chapter 25, "Message Classes," IsParts, page 1303

IsRequest

Description

Use the IsRequest property to determine whether a message is a request message (as opposed to a response message.)

This property returns a Boolean value: true if the message object executing the property is a request message, false otherwise.

This property is read-only.

IsSourceNodeExternal

Description

Use the IsSourceNodeExternal property to determine whether the message came from an internal system or a third-party system.

This property returns a Boolean value: true if the message originated from a third-party system, false otherwise.

This property is read-only.

IsStructure

Description

This property indicates whether the message is rowset-based, that is, based on a rowset, or nonrowset-based that is, based on an XmlDoc. This property returns a Boolean value, true if the message is rowset-based, false otherwise.

This property is read-only.

MessageDetail

Description

This property specifies the message detail, as a string. The message detail can be used to further identify a message.

Note. This property has been deprecated. It is no longer supported.

Name

Description

This property returns the name of the message as a string.

This property is valid with both asynchronous and synchronous messages.

This property is read-only.

NRId

Description

This property returns the non-repudiation ID as a string. This property is populated with a unique string when the message is published.

This property is only valid with messages that use non-repudiation.

This property is read-only.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker Administration, "Setting Up Secure Integration Environments," Implementing Nonrepudiation

OperationName

Description

Use the OperationName property to return the name of the operation associated with the message object executing the property, as a string.

This property is read-only.

OperationVersion

Description

Use the OperationVersion property to determine version for the operation associated with the message object executing the property, as a string.

This property is read-only.

ParentTransactionId

Description

Use the ParentTransactionId property to return the parent transaction ID, that is, the ID that is generated with the original request message from a third-party system.

This property is read-only.

See Also

Chapter 25, "Message Classes," TransactionId, page 1312

PartCount

Syntax

Use the PartCount property to determine the number of parts in a container message. This property is only valid with container messages. You will receive an error if you try to use it on a message that isn't a container message.

This property is read-only.

PubID

Description

This property refers to the publication identifier, which is a number.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class QueueSeqId property instead.

See Chapter 25, "Message Classes," QueueSeqId, page 1308.

It's generated for asynchronous messages when the message is published, and is sequential, based on the number of messages published for that channel. The message instance keys are a subset of the publication and subscription contract keys. This property is one of the message instance keys: the others are ChannelName and PubNodeName.

Note. The PubID property is not populated when the message is published from a notification.

This property is valid only with asynchronous messages.

This property is read-only.

Example

```
&MSG.Publish();
&PUBID = &MSG.PubID;
```

PubNodeName

Description

This property refers to a string that contains the name of the publishing node. It is generated by the system when the message is published. The message instance keys are a subset of the publication and subscription contract keys.

This property is valid for both asynchronous and synchronous messages.

For a synchronous message, this property returns the name of the requesting node.

This property is read-only.

Example

```
&MSG = %IntBroker.GetMessage();
&PUBNODE = &MSG.PubNodeName;
```

QueueName

Description

Use the QueueName property to return the name of the queue associated with the message.

This property is read-only.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Service Operation Queues"

QueueSeqId

Description

Use the QueueSeqId property to return the sequence number of the message in the message queue.

This property is read-only.

ResponseStatus

Description

This property is valid only for the returned response message from a synchronous request message.

This property indicates whether a response was successful or not.

Note. This is not the same as SetStatus, which is used only with the message monitor.

This property returns a numeric value: 0, the response was successful, any other number indicates an error.

This property is read-only.

SegmentCount

Description

This property returns the total number of segments for the message.

This property is read-only.

SegmentsByDatabase

Description

Use this property to specify whether segments are stored in memory while being processed.

If you specify SegmentsByDatabase as false, you can only have the number of segments as specified in PSADMIN (Message Segment From DB). If you specify this property as true, you can have as many segments as you need.

The SegmentsByDatabase property also specifies whether the segments are kept in memory or written to the database when they are received. If you specify true, the segments are automatically written to the database. If you specify false, the segments are held in memory. If you specify true, then cancel out of the program processing the segments, the changes are not committed to the database.

This property is automatically set to true for message segments being processed using a Application Engine program.

This property is read-write.

See Also

Chapter 25, "Message Classes," Message Segments, page 1251

Chapter 25, "Message Classes," SegmentsUnOrder, page 1371

Size

Description

The Size property is the approximate size of the uncompressed XML data generated when the message is published. The availability and meaning of the Size property depends on the message type as follows:

- For rowset-based message parts, the Size property is available and based on the estimated XML size.
- For non-rowset-based messages, the Size property is available and based on the XML data size.
- For container-based messages, the Size property is available on received messages only—that is for OnRequest or OnNotify PoepleCode events.
- For container-based message parts, the Size property is available and based on the estimated XML size.
- For document-based messages, the Size property is available and based on the XML data size.

The size is approximate for the following reasons:

- The maximum size of fields is used except for the case of long text and image fields.
- The active row count is used, without regard to whether the rows have been changed. In a CopyRowsetDelta, rows that are not changed, are not published.
- It doesn't include the size of the XML tags.

This property can be used with the system property %MaxMessageSize in a batch Application Engine program to prevent the application from publishing a message that is too large.

This property is valid for both asynchronous and synchronous messages.

This property is read-only.

Example

```
If &MSG.Size > %MaxMessageSize Then
  &MSG.Publish();
Else
  /*Move more data into the message object */
End-If;
```


See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "System Variables," %MaxMessageSize

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

SubName

Description

This property refers to a string that contains the name of the notification process currently processing the message.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class ActionName property instead.

See Chapter 25, "Message Classes," ActionName, page 1295.

It is available only when GetMessage is used in a notification PeopleCode program.

This property is only valid for asynchronous messages.

This property is read-only.

Example

```
&MSG = %IntBroker.GetMessage();
&SUBNAME = &MSG.SubName
```

SubQueueName

Description

Use the SubQueueName to return or specify the name of the sub-queue associated with the message, as a string.

This property is read-write.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Service Operation Queues"

SubscriptionProcessId

Description

This property returns a string containing the subscription process identifier. This is a unique sequential number.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class TransactionId property instead.

See Chapter 25, "Message Classes," TransactionId, page 1312.

This property is populated only if the "Generate Subscription Process Instance" option is turned on in the Subscription Process definition.

The subscription process identifier corresponds to the subscription process instance, not the message instance. If there are multiple subscription processes for the same message each subscription process will have a different, unique ID.

If the subscription process terminated abnormally, its process instance is lost, and a new one is generated on the next retry (that is, there will be a gap in the sequence.)

This property is valid only for asynchronous messages.

This property is read-only.

Considerations for Using SubscriptionProcessId

Consider the following when using SubscriptionProcessId:

- Using the Run PeopleCode option from Message Subscription does not generate the number. It is generated only if the Application Server is running the PeopleCode.
- If for some reason you have to rerun the subscription, a new number is assigned.
- Because generating the number is optional, your program must contain code to support the option being turned off. Here is a code example from ITEM_SYNC:

  ```
  If All(&MSG.SubscriptionProcessId) Then
    &EIP_CTL_ID = Generate_EIP_CTL_ID(4, &MSG.SubscriptionProcessId);
  Else
    &EIP_CTL_ID = Generate_EIP_CTL_ID(1, "Random");
  End-If;
  ```

- If the flag is off, this property is blank. In this case, you may want to adopt a standard for generating a random number, as shown in the previous example.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

TransactionId

Description

Use the TransactionId property to return the transaction ID for an already published message, as a string. This is a unique transaction identifier for the message.

This property is read-only.

Version

Description

Use the Version property to determine the version of the message executing the property, as a string.

This property is read-only.

IntBroker Class

An IntBroker object is returned from the system variable %IntBroker.

IntBroker Class Methods

In the following section, we discuss the IntBroker class methods. The methods are described in alphabetical order.

Cancel

Syntax

Cancel*(TransactionId,QueueName,DataType, SegmentIndex | TransactionIdArray, QueueNameArray,DataTypeArray, SegmentIndexArray)*

Description

Use the Cancel method to programmatically cancel either a message, or a list of messages, from the message queue, much the same as you can do in the message monitor.

This method is only available when the message has one of the following statuses:

- Edited

- Error
- New
- Retry
- Timeout

If you are specifying an array of messages to be canceled, and any message in the array fails for any reason (for example, you specify a message that doesn't have the correct status) no messages are canceled and the method return value is false.

Parameters

Parameter	***Description***
TransactionId | *TransactionIdArray*	Specify either a single transaction ID as a string, or specify an array of string containing the transaction Ids of the messages you want to cancel.
QueueName | *QueueNameArray*	Specify either a single queue name as a string, or specify an array of string containing the queue names that contain the messages you want to cancel.
DataType | *DataTypeArray*	Specify either a single data type, or an array of string containing the data types. See below for valid data types.
SegmentIndex | *SegmentIndexArray*	Specify either a specific segment, or an array of integer containing the segment numbers you want to cancel.

For the *DataType* or *DataTypeArray* parameters, the valid data types are:

Constant Value	***Description***
%IntBroker_BRK	Cancel the message for the web services gateway.
%IntBroker_PUB	Cancel the message for the publication contract.
%IntBroker_SUB	Cancel the message for the subscription contract.

Returns

A Boolean value: true if the message or messages are canceled successfully, false otherwise.

See Also

Chapter 25, "Message Classes," Resubmit, page 1328

ConnectorRequest

Syntax

ConnectorRequest*(&Message)*

Description

Use the ConnectorRequest method to return a response message from the connector, based on the message passed in.

You would only use this method after you had set the connector information.

Parameters

Parameter	*Description*
&Message	Specify an already instantiated message object to return a response message from the connector.

Returns

A reference to a message object.

Example

```
   Local string &nonXmlData = "<?xml version=""1.0""?><data psnonxml=""yes""><!⇒
[CDATA[" | &encoded | "]]></data>";
   &soapMsg = CreateXmlDoc(&nonXmlData);

   &reqMsg.SetXmlDoc(&soapMsg);

   Local Message &respMsg;

   %This.SetConnectorProperties(&reqMsg, &url);

   &respMsg = %IntBroker.ConnectorRequest(&reqMsg);

   If &respMsg = Null Then /* Throw exception */
      %This.HandleNullSoapResponse(&soapMsg, &url);
   End-If;
```

See Also

Chapter 25, "Message Classes," ConnectorRequestUrl, page 1315

ConnectorRequestUrl

Syntax

`ConnectorRequestUrl`*`(URL)`*

Description

Use the ConnectorRequestUrl method to return the URL for a connector.

You must have already set the connector parameters before executing this command.

Parameters

Parameter	*Description*
URL	Specify an absolute URL as a string.

Returns

A string.

Example

```
/**
 * Get XML Doc from URL
 *
 * @param String - Location
 * @return XmlDoc - retrieved XmlDoc
**/

method getXmlDocumentFromURL
   /+ &location as String +/
   /+ Returns XmlDoc +/
   Local XmlDoc &xmldoc;
   Local string &xmlstr;

   If Substring(LTrim(RTrim(Upper(&location))), 1, 4) = "HTTP" Then
      &xmlstr = %IntBroker.ConnectorRequestUrl(&location);
   Else
      &xmlstr = %This.getXmlContentFromFile(&location);
   End-If;

   &xmldoc = CreateXmlDoc("");

   If &xmldoc.ParseXmlString(&xmlstr) Then
      Return &xmldoc;
   End-If;

end-method;
```

See Also

Chapter 25, "Message Classes," ConnectorRequest, page 1314

DeleteOrphanedSegments

Syntax

`DeleteOrphanedSegments`*`(TransactionId)`*

Description

Use the DeleteOrphanedSegments method to delete segments that might have been orphaned if you were processing message segments using a Application Engine program that had to be restarted. Since each segment is committed to the database, if the application engine program is aborted, these segments need to be deleted from the database.

Parameters

Parameter	*Description*
TransactionId	Specify the transaction ID that identifies the message which contains the segments you want to delete.

Returns

A Boolean value: true if the method completes successfully, false otherwise.

See Also

Chapter 25, "Message Classes," SegmentRestart, page 1285

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Understanding Message Segments

GetArchData

Syntax

`GetArchData`*`(TransactionId, SegmentIndex)`*

Description

Use the GetArchData method to retrieve an archived message from the message queue.

Note. This method shouldn't be used in standard message processing. It should only be used when accessing archived messages.

Parameters

Parameter	***Description***
TransactionId	Specify the transaction ID of the archived message you want to retrieve.
SegmentIndex	Specify the number of the segment of the archived message that you want to retrieve.

Returns

An XML string containing the archived data.

See Also

Chapter 25, "Message Classes," GetArchIBInfoData, page 1317

GetArchIBInfoData

Syntax

`GetArchIBInfoData``(`*`TransactionId,ParentTransactionId`*`)`

Description

Use the GetArchIBInfoData method to retrieve return archived IBInfo data.

Note. This method shouldn't be used in standard message processing. It should only be used when accessing archived messages.

Parameters

Parameter	***Description***
TransactionId	Specify the transaction ID of the archived message you want to retrieve.

Parameter	**Description**
ParentTransactionId	Specify the parent transaction ID, for the message.

Returns

An XML string containing the archived data.

See Also

Chapter 25, "Message Classes," GetArchData, page 1316

GetIBInfoData

Syntax

`GetIBInfoData`*`(TransactionId,DataType)`*

Description

Use the GetIBInfoData to return the specified IBInfo data.

Parameters

Parameter	**Description**
TransactionID	Specify the transaction ID of the message that you want to access.
DataType	Specify the data type of the message you want to access. The valid values are:

Constant Value	**Description**
%IntBroker_BRK	Get the message for the web services gateway.
%IntBroker_PUB	Get the message for the publication contract.
%IntBroker_SUB	Get the message for the subscription contract.

Returns

An XML string containing the message data.

See Also

Chapter 25, "Message Classes," GetArchIBInfoData, page 1317

GetIBTransactionIDforAE

Syntax

GetIBTransactionIDforAE*(OperID,RunCtlID)*

Description

Use this method to get the Integration Broker transaction ID from within an Application Engine program. You can use this transaction ID to instantiate a message object and thereby retrieve the message content data.

Parameters

Parameter	*Description*
OperID	Specifies the operator ID for the Application Engine program as a string.
RunCtlID	Specifies the run control ID for the Application Engine program as a string.

Returns

A string representing the transaction ID.

Example

```
&TransID = %IntBroker.GetIBTransactionIDforAE(&opid, &runid);
&Msg = %IntBroker.GetMessage(&TransID, %IntBroker_SUB);
```

GetMessage

Syntax

GetMessage([*TransactionId,DataType*])

Description

Use the GetMessage method to return a message object.

If you specify a transaction ID and data type, the GetMessage method populates the message object with the data from that message.

If you do not specify any parameters, the GetMessage method doesn't populate the message with data. Instead, it creates a new instance of a message object. In this case, you must use another method, such as GetRowset, to populate the message object. You must always populate the message object with data before running any methods on it.

Parameters

Parameter	***Description***
TransactionId	Specify the transaction ID for the message that you want returned.
DataType	Specify the type of message that you want returned. The valid values are:

Constant Value	***Description***
%IntBroker_BRK	Get the message for the web services gateway.
%IntBroker_PUB	Get the message for the publication contract.
%IntBroker_SUB	Get the message for the subscription contract.

Returns

A reference to a message object if successful, Null if not successful.

Example

The following example returns a populated message object:

```
Local Message &MSG;
Local string &TransactionId;

&TransactionId = "0f3617dl-c6f4-11d9-a4bd-c12cbalbc2f9"
&MSG = %IntBroker.GetMessage(&TransactionId, %IntBroker_BRK);
```

See Also

Chapter 25, "Message Classes," GetRowset, page 1278

Chapter 25, "Message Classes," TransactionId, page 1312

GetMessageErrors

Syntax

GetMessageErrors*(TransactionId)*

Description

Use the GetMessageErrors method to return an array that contains the errors that an application has set for the message, and have been written to the error queue. This method is generally only used with asynchronous messages.

Parameters

Parameter	***Description***
TransactionId	Specify the transaction ID for the message, as a string.

Returns

An array of string. If there are no error messages, the Len property of the array is 0.

Example

The following is an example of using the method for returning web services gateway error messages:

```
&MyErrors = %IntBroker.GetMessageErrors(&TransactionId);
```

See Also

Chapter 25, "Message Classes," SetMessageError, page 1329

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Processing Inbound Errors

GetMsgSchema

Syntax

GetMsgSchema*(MsgName,MsgVersion)*

Description

Use the GetMsgSchema method to access the saved schema for the specified message.

If you specify a message that does not have a saved schema, you receive a null value.

Parameters

Parameter	***Description***
MsgName	Specify the name of the message for which you want to access a schema. You can either specify the name of the message as a string, or preface the message name with the keyword **Message.**
MsgVersion	Specify the message version.

Returns

A string containing the message schema. If you try to get a schema for a message that does not have a saved schema, returns null.

See Also

Chapter 25, "Message Classes," MsgSchemaExists, page 1326

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Building Message Schemas"

GetSyncIBInfoData

Syntax

`GetSyncIBInfoData``(`*`TransactionId,%LogType`*` [, `*`Archive`*`])`

Description

Use the GetSyncIBInfoData method to return a log containing information about synchronous transactions.

Parameters

Parameter	***Description***
TransactionId	Specify the transaction ID of the published message.

Parameter	***Description***
LogType	Specify the type of log you want returned, based on the type of message. See below for the valid values.
Archive	Specify whether to retrieve any archived logs as well. This parameter takes a Boolean value: true to return archived logs, false otherwise. The default value is false.

For the *LogType* parameter, the valid values are:

Constant Value	***Description***
%Sync_RequestOrig	Gets the log for a sync request original data message.
%Sync_RequestTrans	Gets the log for a sync request transformed data message.
%Sync_ResponseOrig	Gets the log for a sync response original data message.
%Sync_ResponseTrans	Gets the log for a sync response transformed data message.

Returns

An XML string containing the log data.

See Also

PeopleTools 8.51 PeopleBook: Integration Broker Service Operations Monitor, "Understanding the Integration Broker Service Operations Monitor"

GetSyncLogData

Syntax

GetSyncLogData*(TransactionId,%LogType* [, *Archive*])

Description

Use the GetSyncLogData method to return a log containing information about the specified synchronous message.

You can use this information for debugging. Using this method, you can obtain the request and response data in a synchronous request, both pre- and post-transformation.

This function is used in the PeopleCode for the Message Monitor.

Parameters

Parameter	***Description***
TransactionId	Specify the transaction ID of the published message.
LogType	Specify the type of log you want returned, based on the type of message. See below for the valid values.
Archive	Specify whether to retrieve any archived logs as well. This parameter takes a Boolean value: true to return archived logs, false otherwise. The default value is false.

For the *LogType* parameter, the valid values are:

Constant Value	***Description***
%Sync_RequestOrig	Gets the log for a sync request original data message.
%Sync_RequestTrans	Gets the log for a sync request transformed data message.
%Sync_ResponseOrig	Gets the log for a sync response original data message.
%Sync_ResponseTrans	Gets the log for a sync response transformed data message.

Returns

An XML string containing the log data.

See Also

PeopleTools 8.51 PeopleBook: Integration Broker Service Operations Monitor, "Understanding the Integration Broker Service Operations Monitor"

InBoundPublish

Syntax

`InBoundPublish`*`(&Message)`*

Description

Use the InBoundPublish method to send an asynchronous message based on the specified message to simulate an inbound asynchronous request from an external node. Although you are sending a message to yourself, it goes through all the inbound message processing that the specified message would.

Prior to call the InBoundPublish method, the Message object needs to populated with the requesting node and external operation name.

See *PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker*, "Sending and Receiving Messages," Simulating Receiving Messages from External Nodes.

Parameters

Parameter	***Description***
&Message	Specify an already instantiated message object that you want to use for the message simulation.

Returns

A Boolean value, true if the message is published successfully, false otherwise.

See Also

Chapter 25, "Message Classes," Publish, page 1326 and Chapter 25, "Message Classes," SyncRequest, page 1332

IsOperationActive

Syntax

```
IsOperationActive(OperationName [,OperationVersion])
```

Description

Use the IsOperationActive method to determine if an operation is active or not.

Parameters

Parameter	***Description***
OperationName	Specify the name of the operation that you want to check the status of.
OperationVersion	Specify the version of the specified operation that you want to check the status of, as a string.

Returns

A Boolean value: true if the operation is active, false otherwise.

MsgSchemaExists

Syntax

`MsgSchemaExists``(`*`MsgName`*`,`*`MsgVersion`*`)`

Description

Use the MsgSchemaExists method to determine if a schema has been created and saved for the specified message.

Parameters

Parameter	*Description*
MsgName	Specify the name of the message for which you want to access a schema. You can either specify the name of the message as a string, or preface the message name with the keyword **Message.**
MsgVersion	Specify the message version.

Returns

A Boolean value: true if the message schema exists, false otherwise.

See Also

Chapter 25, "Message Classes," GetMsgSchema, page 1321

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Building Message Schemas"

Publish

Syntax

`Publish``(`*`&Message`*` [, `*`&ArrayofNodeNames`*`] [,`*`IsEnqueued`*`])`

Description

The Publish method publishes a message to the message queue for the default message version.

This method does not publish the message if the IsOperationActive property for the message is False.

This method publishes a message asynchronously, that is, a reply message is not automatically generated. To publish a message synchronously, use the SyncRequest method.

Note. This method supports nonrowset-based messages only if the data is populated using the SetXmlDoc method.

If the Publish method is called and the message isn't populated (either using SetXmlDoc or one of the rowset methods,) an empty message is published.

The Publish method can be called from any PeopleCode event, but is generally called from an Application Engine PeopleCode step or from a component.

When publishing from a component, you should publish messages only from the SavePostChange event, either from record field or component PeopleCode. This way, if there's an error in your publish code, the data isn't committed to the database. Also, since SavePostChange is the last Save event fired, you avoid locking the database while the other save processing is happening.

However, until the message is published, the tables involved with the component are locked and are not saved. To avoid problems with this, specify the *Deferred_Mode* property as true, so the message is published after the table data has been committed to the database.

Note. If you're publishing a message from within an Application Engine program, the message won't actually be published until the calling program performs a Commit.

Parameters

Parameter	***Description***
&Message	Specify an already instantiated message object to be published.
&ArrayofNodeNames	Specify an already instantiated array of string containing the names of the nodes that you want to publish to.
IsEnqueued	Specify whether the message is to be enqueued or not. This parameter takes a Boolean value: true if the message is to be enqueued, false otherwise. The default value is false.

Returns

None.

Example

```
Local Message &Msg;
Local Rowset &rs;
Local IntBroker &Ib;

&Flight_profile = GetLevel0();

&Msg = CreateMessage(Operation.ASYNC, %IntBroker_Request);

&Msg.CopyRowset(&FlightProfile);

&Ib = %IntBroker;

&Ib.Publish(&Msg);
```

See Also

Chapter 25, "Message Classes," SyncRequest, page 1332

Resubmit

Syntax

Resubmit({*TransactionId,QueueName,DataType*, *SegmentIndex*} | {*TransactionIdArray, QueueNameArray,DataTypeArray*, *SegmentIndexArray*})

Description

Use the Resubmit method to programmatically resubmit either a message, or a list of messages, from the message queue, much the same as you can do in the message monitor.

You may want to use this method after an end-user has finished fixing any errors in the message data, and you want to resubmit the message, rerunning the PeopleCode.

This method is only available when the message has one of the following statuses:

- Canceled
- Edited
- Error
- Timeout

If you are specifying an array of messages to be resubmitted, and any message in the array fails for any reason (for example, you specify a message that doesn't have the correct status) no messages are resubmitted and the method return value is false.

Parameters

Parameter	*Description*
TransactionId | *TransactionIdArray*	Specify either a single transaction ID as a string, or specify an array of string containing the transaction Ids of the messages you want to resubmit.
QueueName | *QueueNameArray*	Specify either a single queue name as a string, or specify an array of string containing the queue names that contain the messages you want to resubmit.
DataType | *DataTypeArray*	Specify either a single data type, or an array of string containing the data types. See below for the valid data type values.
SegmentIndex | *SegmentIndexArray*	Specify either a specific segment, or an array of integer containing the segment numbers you want to resubmit.

For the *DataType* or *DataTypeArray* parameters, the valid data types are:

Constant Value	*Description*
%IntBroker_BRK	Resubmit the message for the web services gateway.
%IntBroker_PUB	Resubmit the message for the publication contract.
%IntBroker_SUB	Resubmit the message for the subscription contract.

Returns

A Boolean value: true if the message or messages are resubmitted successfully, false otherwise.

See Also

Chapter 25, "Message Classes," Cancel, page 1312

SetMessageError

Syntax

```
SetMessageError(TransactionID,MsgSet,MsgNumber,Error_Location,ParamListCounter,
[Param] ...)
```

Description

Use the SetMessageError method to write messages to the error queue. This method is used with asynchronous messages only.

Parameters

Parameter	*Description*
TransactionID	Specify the transaction ID for the message as a string.
MsgSet	Specify the number of the message set that the message associated with the error is associated with.
MsgNumber	Specify the message number that you want to associate with this error.
Error_Location	Specify where the error occurred, or any additional information you want to give about the error. This is a string, and allows 254 characters.
ParamListCounter	An integer value from 0 to 4 specifying the number of *Param* substitution strings to be passed.
Param	Specify up to four *Param* values to be used as substitution strings in the message text. The first *Param* value is used in the first substitution (that is, for %1,) the second is used in the second (that is, for %2), and so on.

Returns

A Boolean value, true if the error message was associated correctly, false otherwise.

Example

The following is an example of setting an error message for the web services gateway:

```
&Rslt = %IntBroker.SetMessageError(&TransactionId, &msgset, &msgnum, "error location", ⇒
4, &parm1, &parm2, &parm3, &parm4);
```

If you are passing just two parameters, then the call would be similar to this:

```
&Rslt = %IntBroker.SetMessageError(&TransactionId, &msgset, &msgnum, "error location", ⇒
2, &parm1, &parm2);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Processing Inbound Errors

Chapter 25, "Message Classes," GetMessageErrors, page 1321

SetStatus

Syntax

SetStatus*(&Message,Status)*

Description

Use the SetStatus method to set the status of a publication or subscription contract, much the same way you do it in the message monitor.

You may want to use this method after an end-user has finished fixing any errors in the message data.

You can set the status of a contract to one of the following statuses:

- Canceled
- Done
- Error
- New

The method is available only when the message instance has one of the following statuses:

- Canceled
- Done
- Error
- New

Parameters

Parameter	***Description***
&Message	Specify an already instantiated message object that you want to set the status for.
Status	Specify the status that you want to set the message to. Valid status are:

Constant Value	***Description***
%Operation_Canceled	Cancel the message.
%Operation_Done	Set the message to done.
%Operation_Error	Set the message to be in error.

Constant Value	*Description*
%Operation_New	Specify the message as new.

Returns

None.

SwitchAsyncEventUserContext

Syntax

SwitchAsyncEventUserContext(*UserID*, *LanguageCode*)

Description

Use the SwitchAsyncEventUserContext method to switch the user context within an Integration Broker asynchronous event.

Parameters

Parameter	*Description*
UserID	Specify the user ID, as a string, to which you want to switch the context.
LanguageCode	Specify the language code, as a string, for the user ID.

Returns

A Boolean value: true if the switch user was successful, false otherwise.

Example

```
&returnValue = %IntBroker.SwitchAsyncEventUserContext("VP1", "ENG");
```

SyncRequest

Syntax

SyncRequest([*&MsgArray*, *&NodeNames*])

Description

Use the SyncRequest method to send multiple messages at the same time. You should only use this message with synchronous messages. You can also use this method to send a single synchronous message at a time, or you can use the SyncRequest Message class method.

If you want to publish messages asynchronously, use the Publish method.

Note. This method supports nonrowset-based messages only if the SetXmlDoc method is used to populate the message prior to using the SyncRequest method.

You must set the thread pool size of the application server on the receiving system to a value greater than 1 (the default) in order to process more than one message at a time.

How many threads you specify determines how many messages are worked on simultaneously. For example, if you have 10 messages in *&MsgArray,* and your thread pool size is 5, then only 5 messages are processed at a time.

The maximum number you can specify for your thread pool size is 10.

Parameters

Parameter	***Description***
&MsgArray	Specify an already instantiated array of messages that contains the messages you want to publish.
&NodeNames	Specify an already instantiated array of string containing the names of the nodes that you want to publish to. The first message in the array of messages is published to the first node, the second message is published to the second node, and so on.

Returns

An array of messages. These are the corresponding messages returned for the published messages. The first message in the returned array corresponds to the first message in the published array, the second message with the second, and so on.

Example

The following is an example of how creating and receiving a series of messages:

```
Local Rowset &FLIGHTPLAN, &FLIGHTPLAN_RETURN;
Local Message &MSG;
Local File &BI_FILE;
Declare Function out_BI_results PeopleCode QE_FLIGHTDATA.QE_ACNUMBER FieldFormula;

Local array of Message &messages;
Local array of Message &return_mesages;

&messages = CreateArrayRept(&MSG, 0);
&return_mesages = CreateArrayRept(&MSG, 0);

&FLIGHT_PROFILE = GetLevel0();
&messages [1] = CreateMessage(OPERATION.QE_FLIGHTPLAN_SYNC);
&messages [1].CopyRowset(&FLIGHT_PROFILE);

&messages [2] = CreateMessage(OPERATION.QE_FLIGHTPLAN_SYNC);
&messages [2].CopyRowsetDelta(&FLIGHT_PROFILE);

&return_mesages = %IntBroker.SyncRequest(&messages);

&FLIGHTPLAN_RETURN = &return_mesages [1].GetRowset();
&temp = &return_mesages [1].GenXMLString();

out_BI_results(&temp);

&FLIGHTPLAN_RETURN = &return_mesages [2].GetRowset();
&temp = &return_mesages [2].GenXMLString();

out_BI_results(&temp);
```

See Also

Chapter 25, "Message Classes," SyncRequest, page 1291

Update

Syntax

Update*(&Message)*

Description

Use the Update method to update a message in the message queue with the specified message object.

Parameters

Parameter	*Description*
&Message	Specify an already instantiated and populated message object to be used to update an existing message in the message queue.

Returns

A Boolean value, true if the message is updated successfully, false otherwise.

See Also

Chapter 25, "Message Classes," UpdateXmlDoc, page 1335

UpdateXmlDoc

Syntax

`UpdateXmlDoc``(`*`&XmlDoc,TransactionId,DataType`*`)`

Description

Use the UpdateXmlDoc method to update a message in the message queue with the specified message data.

Note. This method shouldn't be used in a subscription program.

Parameters

Parameter	***Description***
&XmlDoc	Specify an already instantiated and populated XmlDoc object.
TransactionId	Specify the transaction ID of the message you want to update.
DataType	Specify the data type of the message that you want to update. Valid values are:

Constant Value	***Description***
%IntBroker_BRK	Update the message for the web services gateway.
%IntBroker_PUB	Update the message for the publication contract.
%IntBroker_SUB	Update the message for the subscription contract.

Returns

A Boolean value, true if the message was updated successfully, false otherwise.

See Also

Chapter 25, "Message Classes," Update, page 1334

IBInfo Class

An IBInfo object is returned from the IBInfo message class property.

See Also

Chapter 25, "Message Classes," IBInfo, page 1299

IBInfo Class Methods

In this section, we discuss the IBInfo class methods. The methods are discussed in alphabetical order.

All methods work with both asynchronous as well as synchronous messages unless specified

AddAEAttribute

Syntax

`AddAEAttribute``(`*`Name, value`*`)`

Description

Use this method to add an attribute as a name-value pair to the Message object to be passed back to the response application class defined on the Application Engine handler.

Parameters

Parameter	***Description***
Name	Specifies the attribute name as a string.
value	Specifies the attribute value as a string.

Returns

A Boolean value: True if the attribute was added successfully, False otherwise.

Example

```
/* Add the name value pair data you want to pass to the response app class */
/* in the AE program */

&b = &MSG.IBInfo.AddAEAttribute("sail name", "NorthWave");
If &b <> True Then
   MessageBox(0, "", 0, 0, "error adding ae attribute");
End-If;

&b = &MSG.IBInfo.AddAEAttribute("size", "4.2");
If &b <> True Then
   MessageBox(0, "", 0, 0, "error adding ae attribute");
End-If;

&b = &MSG.IBInfo.AddAEAttribute("type", "Surflite");
If &b <> True Then
   MessageBox(0, "", 0, 0, "error adding ae attribute");
End-If;

/* Need to call this method for the attributes to be saved and transferred */
/* correctly */
&b = &MSG.IBInfo.InsertAEResponseAttributes();
If &b <> True Then
   MessageBox(0, "", 0, 0, "error inserting AE response attributes");
End-If;

/* ****************************************************************** */
/* In the response application class to retrieve the name-value pairs      */

For &i = 1 To &MSG.IBInfo.GetNumberOfAEAttributes()

   &name = &MSG.IBInfo.GetAEAttributeName(&i);
   &value = &MSG.IBInfo.GetAEAttributeValue(&i);

End-For;
```

See Also

Chapter 25, "Message Classes," ClearAEAttributes, page 1341; Chapter 25, "Message Classes," DeleteAEAttribute, page 1344; Chapter 25, "Message Classes," GetAEAttributeName, page 1347; Chapter 25, "Message Classes," GetAEAttributeValue, page 1347; Chapter 25, "Message Classes," GetNumberofAEAttributes, page 1354; Chapter 25, "Message Classes," GetTransactionIDforAE, page 1356 and Chapter 25, "Message Classes," InsertAEResponseAttributes, page 1357

AddAttachment

Syntax

`AddAttachment`*`(Path)`*

Description

Use the AddAttachment method to add an attachment to a message.

Parameters

Parameter	***Description***
Path	Specify an absolute path, including the file name and type, as a string, of the file you want to attach to the message.

Returns

A string containing a content ID, which can be used for accessing the attachment with other methods, such as DeleteAttachment or SetAttachmentProperty.

Example

The following example shows sending an attachment with an asynchronous message:


```
Local Message &MSG;
Local Rowset &FLIGHT_PROFILE;
Local String &Attachment_Id;
Local Boolean &Test;
Local IntBroker &IntBroker;

QE_FLIGHTDATA.QE_ACNUMBER.Value = QE_FLIGHTDATA.QE_ACNUMBER + 1;

&FLIGHT_PROFILE = GetLevel0();

&MSG = CreateMessage(Operation.ASYNC_RR);

&Attachment_Id = &MSG.IBInfo. AddAttachment("C:\\temp\\MyFile.txt");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Encoding,⇒
 "UTF-8");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Base,⇒
 "Standard");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Disposition,⇒
 "Pending");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Language,⇒
 "English");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Description,⇒
 "Parts data");

&MSG.CopyRowset(&FLIGHT_PROFILE);

&IntBroker = %IntBroker

&IntBroker.Publish(&MSG);
```

See Also

Chapter 25, "Message Classes," ClearAttachments, page 1342; Chapter 25, "Message Classes," DeleteAttachment, page 1344 and Chapter 25, "Message Classes," SetAttachmentProperty, page 1360

AddAttribute

Syntax

AddAttribute*(name, value)*

Description

Use this method to add an attribute to an IBInfo object.

Note. This method can be used for content-based routing only, typically in your implementation of the IRouter.OnRouteSend or IRouter.OnRouteReceive methods.

The maximum length of the attribute name is 30 characters. The maximum length of the attribute value is 254 characters.

Parameters

Parameter	*Description*
name	Specifies the attribute name as a string.
value	Specifies the attribute value as a string.

Returns

A Boolean value: true if the addition of the attribute was successful, false otherwise.

Example

```
&bRet = &MSG.IBInfo.AddAttribute("employeeID", "123456");
```

See Also

Chapter 25, "Message Classes," Content-Based Routing, page 1254

AddContainerAttribute

Syntax

AddContainerAttribute*(Name,Value)*

Description

Use this method to add a container attribute by specifying an attribute name-value pair.

You can add attributes to container messages that contain rowset-based message parts to provide integration partners with data and information, without adding the information to the message definition.

Parameters

Parameter	*Description*
Name	Specifies the container attribute name as a string.
Value	Specifies the container attribute value as a string.

Returns

A Boolean value: True if the container attribute was added successfully, False otherwise..

Example

```
Local Message &MSG;
Local Rowset &RS;
Local boolean &Bo;

&RS = GetLevel0();
&MSG = CreateMessage(Operation.QE_CONT_ATTRB);

&Bo = &MSG.IBInfo.AddContainerAttribute("Test1", "1");
&Bo = &MSG.IBInfo.AddContainerAttribute("Test2", "10");
&Bo = &MSG.IBInfo.AddContainerAttribute("Test3", "100");

&MSG.CopyPartRowset(1, &RS);

%IntBroker.Publish(&MSG);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Managing Container Messages

Chapter 25, "Message Classes," ClearContainerAttributes, page 1343; Chapter 25, "Message Classes," DeleteContainerAttribute, page 1346; Chapter 25, "Message Classes," GetContainerAttributeName, page 1352; Chapter 25, "Message Classes," GetContainerAttributeValue, page 1353 and Chapter 25, "Message Classes," GetNumberOfContainerAttributes, page 1355

ClearAEAttributes

Syntax

ClearAEAttributes()

Description

Use this method to delete all Application Engine handler attributes from the Message object.

Parameters

None.

Returns

None.

See Also

Chapter 25, "Message Classes," AddAEAttribute, page 1336 and Chapter 25, "Message Classes," DeleteAEAttribute, page 1344

ClearAttachments

Syntax

`ClearAttachments``()`

Description

Use the ClearAttachments method to clear all attachments. If you want to delete a particular attachment, use DeleteAttachment instead.

Parameters

None.

Returns

None.

See Also

Chapter 25, "Message Classes," AddAttachment, page 1338 and Chapter 25, "Message Classes," DeleteAttachment, page 1344

ClearAttributes

Syntax

`ClearAttributes``()`

Description

Use this method to delete all attributes from an IBInfo object.

Note. This method can be used for content-based routing only, typically in your implementation of the IRouter.OnRouteSend or IRouter.OnRouteReceive methods.

Parameters

None.

Returns

None.

Example

```
&MSG.IBInfo.ClearAttributes();
```

See Also

Chapter 25, "Message Classes," Content-Based Routing, page 1254

ClearContainerAttributes

Syntax

`ClearContainerAttributes``()`

Description

Use this method to delete all container attributes in the IBInfo object.

Parameters

None

Returns

None

Example

```
&MSG.IBInfo.ClearContainerAttributes();
```

See Also

Chapter 25, "Message Classes," AddContainerAttribute, page 1340 and Chapter 25, "Message Classes," DeleteContainerAttribute, page 1346

DeleteAEAttribute

Syntax

DeleteAEAttribute*(Name)*

Description

Use this method to delete the Application Engine handler attribute specified by attribute name from the Message object.

Parameters

Parameter	***Description***
Name	Specifies the name of the attribute as a string.

Returns

A Boolean value: True if the deletion was successful, False otherwise.

See Also

Chapter 25, "Message Classes," AddAEAttribute, page 1336; Chapter 25, "Message Classes," ClearAEAttributes, page 1341 and Chapter 25, "Message Classes," GetAEAttributeName, page 1347

DeleteAttachment

Syntax

DeleteAttachment*(Index | Content_ID)*

Description

Use the DeleteAttachment method to remove the specified attachment from the message. You can either specify the number of the attachment, or the content ID associated with the attachment (generated when the attachment was added to the message with AddAttachment.)

If you want to clear all attachments, instead of a particular one, use the ClearAttachments methods instead.

Parameters

Parameter	***Description***
Index | Content_ID	Specify either the number of the attachment, or the content ID associated with the attachment, for the attachment you want to delete.

Returns

A Boolean value: true if the attachment was deleted successfully, false otherwise.

See Also

Chapter 25, "Message Classes," AddAttachment, page 1338 and Chapter 25, "Message Classes," ClearAttachments, page 1342

DeleteAttribute

Syntax

`DeleteAttribute`*`(name)`*

Description

Use this method to delete an attribute from an IBInfo object by specifying the attribute's name as a string.

Note. This method can be used for content-based routing only, typically in your implementation of the IRouter.OnRouteSend or IRouter.OnRouteReceive methods.

Parameters

Parameter	***Description***
name	Specifies the attribute by name as a string.

Returns

A Boolean value: true if the deletion of the attribute was successful, false otherwise.

Example

```
&bRet = &MSG.IBInfo.DeleteAttribute("employeeID");
```

See Also

Chapter 25, "Message Classes," Content-Based Routing, page 1254

Chapter 25, "Message Classes," GetAttributeName, page 1351

DeleteContainerAttribute

Syntax

`DeleteContainerAttribute`*`(Name)`*

Description

Use this method to delete a container attribute based on the attribute name.

Parameters

Parameter	*Description*
Name	Specifies the container attribute name as a string.

Returns

A Boolean value: True if the deletion was successful, False otherwise.

Example

```
&Ret = &MSG.IBInfo.DeleteContainerAttribute("MyAttribute");
```

See Also

Chapter 25, "Message Classes," AddContainerAttribute, page 1340; Chapter 25, "Message Classes," ClearContainerAttributes, page 1343 and Chapter 25, "Message Classes," GetContainerAttributeName, page 1352

GetAEAttributeName

Syntax

`GetAEAttributeName(nIndex)`

Description

Use this method to return the name of the *nth* Application Engine handler attribute from the Message object. For example, the response application class can use this method to retrieve the attribute name.

Parameters

Parameter	*Description*
nIndex	An integer specifying which attribute in the Message object.

Returns

A string populated with the attribute name.

See Also

Chapter 25, "Message Classes," AddAEAttribute, page 1336; Chapter 25, "Message Classes," GetAEAttributeValue, page 1347 and Chapter 25, "Message Classes," GetNumberofAEAttributes, page 1354

GetAEAttributeValue

Syntax

`GetAEAttributeValue(nIndex)`

Description

Use this method to return the value of the *nth* Application Engine handler attribute from the Message object. For example, the response application class can use this method to retrieve the attribute value.

Parameters

Parameter	Description
nIndex	An integer specifying which attribute in the Message object.

Returns

A string populated with the attribute value.

See Also

Chapter 25, "Message Classes," AddAEAttribute, page 1336; Chapter 25, "Message Classes," GetAEAttributeName, page 1347 and Chapter 25, "Message Classes," GetNumberofAEAttributes, page 1354

GetAttachmentContentID

Syntax

`GetAttachmentContentID``(Index)`

Description

Use the GetAttachmentContentID to return the content ID for the specified attachment. The content ID is associated with an attachment when it is added to a message using AddAttachment.

You can use the content ID with other methods, such as AddAttachmentProperty and DeleteAttachment.

Parameters

Parameter	Description
Index	Specify the number of the attachment that you want to access the content ID for.

Returns

A string containing the content ID.

See Also

Chapter 25, "Message Classes," AddAttachment, page 1338 and Chapter 25, "Message Classes," GetAttachmentProperty, page 1349

GetAttachmentProperty

Syntax

GetAttachmentProperty(*Content_ID*,*Property_Type*)

Description

Use the GetAttachmentProperty method to return the value of an attachment property.

Parameters

Parameter	***Description***
Content_ID	Specify the content ID of the attachment that you want to access, as a string.
Property_Type	Specify the type of property that you want to access. Valid values are:

Constant Value	***Description***
%Attachment_Base	Specifies the base name of the attachment, that is, if the actual document name you want is different, such as if it is zipped or otherwise compressed as part of other documents.
%Attachment_Description	Specifies the description of the attachment.
%Attachment_Disposition	Specifies the disposition of the attachment, whether it's displayed inline or as an attachment.
%Attachment_Encoding	Specify the encoding of the attachment. For example, `Content-type: text/plain` or `charset=Big5`.
%Attachment_Language	Specify the language used in the attachment, as a string.
%Attachment_Location	Specify an additional location for the attachment.
%Attachment_Type	Specify the attachment type.
%Attachment_URL	Specifies the URL for the attachment. The URL must be an absolute URL.

Returns

A string containing the value of the specified property.

Example

The following example processes an attachment from a notification PeopleCode program:

```
Import PS_PT:Integration:INotificationHandler;

Class FLIGHTPROFILE implements PS_PT:Integration:INotificationHandler
   method FLIGHTPROFILE();
   method OnNotify(&MSG As Message);

end-class;

/* Constructor */
method FLIGHTPROFILE
   %Super = create PS_PT:Integration:INotificationHandler();
end-method;

method OnNotify
   /+ &MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/

Local rowset &RS;
Local integer &Count;
Local string &Attachment_Id;
Local array of array of string &Result;

&RS = &MSG.GetRowset();

&Count = &MSG.IBInfo.NumberOfAttachments;
If &Count > 0 Then

   For &I = 1 to &Count
      &Attachment_ID = &MSG.IBInfo.GetAttachmentContentID(&I);
      &Result[&I][1] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Encoding);
      &Result[&I][2] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Type);
      &Result[&I][3] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_URL);
      &Result[&I][4] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Base);
      &Result[&I][5] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Location);
      &Result[&I][6] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Disposition);
      &Result[&I][7] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Description);

   End-For;

End-if;

/* process data from message */

end-method;
```

See Also

Chapter 25, "Message Classes," SetAttachmentProperty, page 1360

GetAttributeName

Syntax

`GetAttributeName`*`(nIndex)`*

Description

Use this method to return the attribute name for the *nth* attribute of the IBInfo object as a string.

Note. This method can be used for content-based routing only, typically in your implementation of the IRouter.OnRouteSend or IRouter.OnRouteReceive methods.

Parameters

Parameter	*Description*
nIndex	An integer specifying which attribute in the IBInfo object.

Returns

A string populated with the attribute name.

Example

```
&AttrName = &MSG.IBInfo.GetAttributeName(&i);
```

See Also

Chapter 25, "Message Classes," Content-Based Routing, page 1254

Chapter 25, "Message Classes," GetAttributeValue, page 1352 and Chapter 25, "Message Classes," GetNumberOfAttributes, page 1354

GetAttributeValue

Syntax

`GetAttributeValue``(`*`nIndex`*`)`

Description

Use this method to return the attribute value for the *nth* attribute of the IBInfo object as a string.

Note. This method can be used for content-based routing only, typically in your implementation of the IRouter.OnRouteSend or IRouter.OnRouteReceive methods.

Parameters

Parameter	*Description*
nIndex	An integer specifying which attribute in the IBInfo object.

Returns

A string populated with the attribute value.

Example

```
&AttrValue = &MSG.IBInfo.GetAttributeValue(&i);
```

See Also

Chapter 25, "Message Classes," Content-Based Routing, page 1254

Chapter 25, "Message Classes," GetAttributeName, page 1351 and Chapter 25, "Message Classes," GetNumberOfAttributes, page 1354

GetContainerAttributeName

Syntax

`GetContainerAttributeName``(`*`nIndex`*`)`

Description

Use this method to return the attribute name for the *n*th container attribute of the IBInfo object as a string.

Parameters

Parameter	*Description*
nIndex	An integer specifying which container attribute in the IBInfo object.

Returns

A string populated with the container attribute name.

See Also

Chapter 25, "Message Classes," AddContainerAttribute, page 1340; Chapter 25, "Message Classes," GetContainerAttributeValue, page 1353 and Chapter 25, "Message Classes," GetNumberOfContainerAttributes, page 1355

GetContainerAttributeValue

Syntax

`GetContainerAttributeValue`*`(nIndex)`*

Description

Use this method to return the attribute value for the *n*th container attribute of the IBInfo object as a string.

Parameters

Parameter	*Description*
nIndex	An integer specifying which container attribute in the IBInfo object.

Returns

A string populated with the container attribute value.

See Also

Chapter 25, "Message Classes," AddContainerAttribute, page 1340; Chapter 25, "Message Classes," GetContainerAttributeName, page 1352 and Chapter 25, "Message Classes," GetNumberOfContainerAttributes, page 1355

GetNumberofAEAttributes

Syntax

`GetNumberofAEAttributes``()`

Description

Use this method to return the number of Application Engine handler attributes in the Message object. For example, the response application class can use this method to retrieve the number of attributes before retrieving the attribute name-value pairs.

Parameters

None.

Returns

An integer representing the number of Application Engine handler attributes.

See Also

Chapter 25, "Message Classes," AddAEAttribute, page 1336; Chapter 25, "Message Classes," GetAEAttributeName, page 1347 and Chapter 25, "Message Classes," GetAEAttributeValue, page 1347

GetNumberOfAttributes

Syntax

`GetNumberOfAttributes``()`

Description

Use this method to get the number of attributes for the IBInfo object as an integer.

Note. This method can be used for content-based routing only, typically in your implementation of the IRouter.OnRouteSend or IRouter.OnRouteReceive methods.

Parameters

None.

Returns

An integer representing the number of attributes for the IBInfo object.

Example

```
For &i = 1 To &MSG.IBInfo.GetNumberOfAttributes()
   &AttrName = &MSG.IBInfo.GetAttributeName(&i);
   &AttrValue = &MSG.IBInfo.GetAttributeValue(&i);
End-For;
```

See Also

Chapter 25, "Message Classes," Content-Based Routing, page 1254

Chapter 25, "Message Classes," GetAttributeName, page 1351 and Chapter 25, "Message Classes," GetAttributeValue, page 1352

GetNumberOfContainerAttributes

Syntax

GetNumberOfContainerAttributes()

Description

Use this method to return the number of container attributes in the IBInfo object.

Parameters

None

Returns

An integer representing the number of container attributes.

Example

```
&MSG = CreateMessage(Operation.FLIGHTPLAN);

&ret = &MSG.IBInfo.AddContainerAttribute("MESSAGE_STATUS", "good");
&ret = &MSG.IBInfo.AddContainerAttribute("MyAttribute", "abcd");

/* When attempting to read these attributes within an IB event (OnNotify,  */
/* OnRequest etc.) one must first get a part rowset, this will load up the */
/* attributes into the message object from the xml. Here is an example of  */
/* how to read the attributes from a message.                              */

&RS = &MSG.GetPartRowset(1);
&index = &MSG.Ibinfo.GetNumberOfContainerAttributes();

For &i = 1 To &index

   &attrName = &MSG.Ibinfo.GetContainerAttributeName(&i);
   &attrValue = &MSG.Ibinfo.GetContainerAttributeValue(&i);

End-For;
```

See Also

Chapter 25, "Message Classes," AddContainerAttribute, page 1340; Chapter 25, "Message Classes," GetContainerAttributeName, page 1352 and Chapter 25, "Message Classes," GetContainerAttributeValue, page 1353

GetTransactionIDforAE

Syntax

GetTransactionIDforAE()

Description

Use this method to get the transaction ID from within Application Engine program.

Parameters

None.

Returns

A number representing the transaction ID from within Application Engine program.

See Also

Chapter 25, "Message Classes," AddAEAttribute, page 1336

InsertAEResponseAttributes

Syntax

`InsertAEResponseAttributes``()`

Description

Use this method to save and transfer the Application Engine handler attributes to be read by the response application class.

Parameters

None.

Returns

A Boolean value: True if the save and insertion were successful, False otherwise.

See Also

Chapter 25, "Message Classes," AddAEAttribute, page 1336

LoadConnectorProp

Syntax

`LoadConnectorProp``(`*`ConnectorName`*`)`

Description

Use the LoadConnectorProp method to load connector properties to the specified connector. The properties are contained in the message executing the method.

Note. Use this method in the message OnSend event.

Parameters

Parameter	*Description*
ConnectorName	Specify the name of the connector that you want to load properties for from the message.

Returns

A Boolean value: true if properties are loaded successfully, false otherwise.

Example

```
LOCAL MESSAGE &MSG;
&MSG = %IntBroker.GetMessage();
&Rowset = &MSG.GetRowset();
&MSG.IBInfo.LoadConnectorProp("HTTP TargetConnector");
/* add connector properties */
&MSG.IBInfo.ConnectorOverride= true;
ReturnToServer(&MSG);
```

See Also

Chapter 25, "Message Classes," ConnectorOverride, page 1363

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReturnToServer

LoadConnectorPropFromNode

Syntax

`LoadConnectorPropFromNode`*`(NodeName)`*

Description

Use the LoadConnectorPropFromNode method to load connector properties into the message executing the method. Then you can use the LoadConnectorProp method to load the specified connector with the properties.

Note. Use this method in the message OnSend event.

Parameters

Parameter	*Description*
NodeName	Specify the node that contains the connector properties you want to use. You can either specify the node name as a string, or prefix the node name with the reserved word **Node.**

Returns

A Boolean value: true if the properties are loaded successfully, false otherwise.

See Also

Chapter 25, "Message Classes," ConnectorOverride, page 1363

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReturnToServer

LoadConnectorPropFromRouting

Syntax

`LoadConnectorPropFromRouting``(`*`RoutingDefnName`*`)`

Description

Use the LoadConnectorPropFromRouting method to load connector properties into the message executing the method. Then you can use the LoadConnectorProp method to load the specified connector with the properties.

Parameters

Parameter	*Description*
RoutingDefnName	Specify the routing definition name that contains the connector properties you want to use, as a string.

Returns

A Boolean value: true if the method completes successfully, false otherwise.

See Also

Chapter 25, "Message Classes," LoadConnectorProp, page 1357 and Chapter 25, "Message Classes," LoadConnectorPropFromNode, page 1358

LoadConnectorPropFromTrx

Syntax

`LoadConnectorPropFromTrx``(`*`NodeName,TransactionType, MsgName,MsgVersion`*`)`

Description

Note. This method is no longer supported.

SetAttachmentProperty

Syntax

`SetAttachmentProperty``(`*`Content_ID,Property_Type, Value`*`)`

Description

Use the SetAttachmentProperty to specify the value and type of a property associated with an attachment.

Parameters

Parameter	***Description***
Content_ID	Specify the content ID associated with the attachment.
Property_Type	Specify the type of property. See below for the valid values.
Value	Specify the value associated with this property.

For the *Property_Type* parameter, the valid values are:

Constant Value	***Description***
%Attachment_Encoding	Specify the encoding type.
%Attachment_Type	Specify the attachment type.

Constant Value	***Description***
%Attachment_URL	Specifies the URL for the attachment. The URL must be an absolute URL.
%Attachment_Base	Specify the attachment base.
%Attachment_Location	Specify an additional location for the attachment.
%Attachment_Disposition	Specifies the disposition of the attachment, whether it's displayed inline or as an attachment.
%Attachment_Description	Specifies the description of the attachment.

Returns

A Boolean value: true if the property is set successfully, false otherwise.

Example

The following example shows sending an attachment with an asynchronous message:

```
Local Message &MSG;
Local Rowset &FLIGHT_PROFILE;
Local String &Attachment_Id;
Local Boolean &Test;
Local IntBroker &IntBroker;

QE_FLIGHTDATA.QE_ACNUMBER.Value = QE_FLIGHTDATA.QE_ACNUMBER + 1;

&FLIGHT_PROFILE = GetLevel0();

&MSG = CreateMessage(Operation.ASYNC_RR);

&Attachment_Id = &MSG.IBInfo. AddAttachment("C:\\temp\\MyFile.txt");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Encoding,⇒
 "UTF-8");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Base,⇒
 "Standard");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Disposition,⇒
 "Pending");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Language,⇒
 "English");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Description,⇒
 "Parts data");

&MSG.CopyRowset(&FLIGHT_PROFILE);

&IntBroker = %IntBroker

&IntBroker.Publish(&MSG);
```

See Also

Chapter 25, "Message Classes," GetAttachmentProperty, page 1349

IBInfo Class Properties

In this section, we discuss the IBInfo class properties. The properties are discussed in alphabetical order.

AppServerDomain

Description

This property can be used to set the application server domain for the connector, as a string.

This property is read-write.

CompressionOverride

Description

This property can be used to set a compression override for the transaction. This property takes three system constants to set the compression override:

Constant Value	***Description***
%IntBroker_Compress	Turns compression on for this transaction.
%IntBroker_UnCompress	Turns compression off for this transaction.
%IntBroker_Compress_Reset	Resets compression to use the threshold specified by PSADMIN.

The integration engine compresses and base64-encodes messages destined for the PeopleSoft listening connector on its local integration gateway.

Asynchronous messages are always compressed and base64 encoded when sent to the integration gateway.

For synchronous messages, in PSADMIN you can set a threshold message size above which messages are compressed. With the CompressionOverride property, you can override the message compression setting specified in PSADMIN at the transaction level.

Warning! Turning compression off can negatively impact system performance when transporting synchronous messages greater than 1 MB. As a result, you should turn off compression only during integration development and testing.

Note. This property does not affect the compression of messages that the integration gateway sends using its target connectors.

This property is read-write.

Example

```
&MSG.IBInfo.CompressionOverride = %IntBroker_UnCompress;
```

ConnectorOverride

Description

This property specifies whether the connector override is specified for the transaction. This property takes a Boolean value: true, override the connection properties, false otherwise.

This property is read-write.

ConversationID

Description

This property returns the conversation ID associated with the request/response message transaction.

This property is read-write.

DeliveryMode

Description

This property sets or returns the delivery mode override for the connector as an integer. This property takes one of three system constants to set the delivery mode override:

Constant Value	***Description***
%IB_DM_BestEffort	Sets the delivery mode to best effort. If the delivery mode is set to best effort, then only one attempt is made to send the message to the destination; there is no attempt to retry the message.
%IB_DM_Guarantee	Sets the delivery mode to guaranteed. This mode automatically invokes retry logic to ensure that the message is successfully sent to a destination. This is the default delivery mode.
%IB_DM_Reset	Resets the delivery mode to the value defined outside the PeopleCode program.

This property is read-write.

Example

```
&int = &MSG.IBInfo.DeliveryMode = %IB_DM_BestEffort;
```

DestinationNode

Description

This property returns the name of the destination node that the request was sent to, as a string.

This property is read-only.

ExternalMessageID

Description

This property returns the external ID of the message. This property is used in testing, and to resolve duplicate message issues from third-party systems.

This property is read-only.

Example

```
&str = &MSG.IBInfo.ExternalMessageID;
```

ExternalOperationName

Description

This property returns the external operation name of the message. This property is used in testing, and to resolve duplicate message issues from third-party systems.

This property is read-write.

ExternalUserName

Description

This property returns the external user name associated with the message. This property is used in testing, and to resolve duplicate message issues from third-party systems.

This property is read-write.

ExternalUserPassword

Description

This property returns the external user password associated with the message. This property is used in testing, and to resolve duplicate message issues from third-party systems.

This property is read-write.

FinalDestinationNode

Description

When the message is passed across several nodes, this property specifies the ultimate target of the message, as a string.

This property is read-only.

FuturePublicationDateTime

Description

Use this property to specify when, as a datetime, an actual publish of the transaction is to occur.

This property is for use with asynchronous transactions. If a null value or an invalid future date and time is specified, the publish will occur immediately.

This property is read-write.

HTTPSessionId

Description

Use the HTTPSessionId property to specify the HTTP session ID, as a string.

This property is read-write.

IBConnectorInfo

Description

This property returns a reference to a IBConnectorInfo collection object.

This property is read-only.

InReplyToID

Description

Use the InReplyToID property to specify the reply to ID contained in the message.

This property is read-write.

MessageChannel

Description

This property references the name of the channel associated with the message definition, as a string.

Note. This property has been deprecated and remains for backward compatibility only. Use the IBInfo class MessageQueue property instead.

This property is set in when the message is created.

This property is read-only.

See Also

Chapter 25, "Message Classes," MessageQueue, page 1367

See Also

Chapter 25, "Message Classes," ChannelName, page 1295

MessageName

Description

This property returns the name of the message, as a string.

This property is read-only.

See Also

Chapter 25, "Message Classes," Name, page 1305

MessageQueue

Description

This property returns the name of the queue associated with the message, as a string.

This property is read-only.

MessageType

Description

This property returns the type of the message, as a string.

Note. This property has been deprecated and remains for backward compatibility only. Use the IBInfo class OperationType property instead.

See Chapter 25, "Message Classes," OperationType, page 1368.

Valid types are:

Value	***Description***
Sync	Indicates that the message is synchronous.
Async	Indicates that the message is asynchronous.
Ping	Indicates that the message is used to test the application server to make sure it is available and accepting requests.

This property is read-only.

MessageVersion

Description

This property returns the message version as a number.

This property is read-only.

NodeDN

Description

For incoming requests, this property gives the distinguished name (DN) extracted from the certificate authentication process, as a string.

This property is read-write.

NonRepudiationID

Description

This property returns the non-repudiation ID as a string. This property is populated with a unique string when the message is published.

This property is only valid with messages that use non-repudiation.

This property is read-only.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker Administration, "Setting Up Secure Integration Environments," Implementing Nonrepudiation

NumberOfAttachments

Description

This property returns the number of attachments associated with a message.

This property is read-only.

See Also

Chapter 25, "Message Classes," AddAttachment, page 1338

OperationType

Description

This property returns the type of the operation, as a string.

This property is read-only.

OperationVersion

Description

This property returns the version of the operation, as a string.

This property is read-only.

OrigNode

Description

For requests that cross multiple nodes, this property identifies the node that initiated the request as a string.

The OrigNode property returns the originating node of the message. If the message is not going across nodes, the OrigNode and SoureNode properties return the same value. However, if the message is going across nodes, the source node is the node that most recently published the message.

For example, if A publishes to B, then B publishes the message to C, from C's perspective, A is the original node and B is the source node.

This property is read-only.

See Also

Chapter 25, "Message Classes," SourceNode, page 1371

OrigProcess

Description

This property returns the name of the process where the publish originated, as a string. For example, a message published from the Inventory definitions page would have a process name of INVENTORY DEFIN.

This property is read-only.

OrigTimeStamp

Description

This property returns the time stamp that corresponds to the time that the request was created, as a string. For requests that cross nodes, this is the time that the first request was created.

This property is read-only.

OrigUser

Description

This property returns the user ID login where the message was initially generated, as a string.

This property is read-only.

PublicationID

Description

This property returns the unique identifier for the message, as a string.

Note. This property has been deprecated. It is no longer supported.

RequestingNodeName

Description

This property returns the name of the node making the request, as a string.

This property is read-write.

RequestingNodeDescription

Description

This property returns the description of the node making the request, as a string.

This property is read-write.

ResponseAsAttachment

Description

Use the ResponseAsAttachment property to specify whether the response should be returned as an attachment or inline.

This property is read-write.

SegmentsUnOrder

Description

The SegmentUnOrder property is only applicable for asynchronous messages. If you specify the SegmentUnOrder property as true, the receiving node processes the segments in parallel.

This property is read-write.

See Also

Chapter 25, "Message Classes," SegmentsByDatabase, page 1308

Chapter 25, "Message Classes," Message Segments, page 1251

SourceNode

Description

This property returns the name of the publishing node as a string.

The OrigNode property returns the originating node of the message. If the message is not going across nodes, the OrigNode and SoureNode properties return the same value. However, if the message is going across nodes, the source node is the node that most recently published the message.

For example, if A publishes to B, then B publishes the message to C, from C's perspective, A is the original node and B is the source node.

This property is read-only.

See Also

Chapter 25, "Message Classes," OrigNode, page 1369

SyncServiceTimeout

Description

This property takes a time (in seconds). This time overrides the default HTTP timeout that is used for all requests. If you set this property, you can use this to read back the time, that is, set the time before the SyncRequest is executed, then see when it is changed in an implementation of the OnSend method.

Note. This property is only for synchronous requests.

Generally, you use SyncServiceTimeout to dynamically set the timeout value for a specific transaction. The http header file is modified to take this new parameter. In addition this value is sent to the gateway to be used for the http timeout. Use this so that a long running transaction will not timeout. In addition, you don't have to wait through the default period for a specific transaction to timeout.

The following is a typical example of using this property:

```
&MSG.SetXmlDoc(&xmlReq);
&MSG.IBInfo.LoadConnectorPropFromNode(Node.EAI)
&MSG.IBInfo.SyncServiceTimeout = 360000;
&MSG.IBInfo.ConnectorOverride = true;
&MSG_Resp = SyncRequest(&MSG, Node.EAI);
&xmlResponseDoc = &MSG.GetXmlDoc();
```

Setting the XML directly is not valid. You need to use the message object to set the value. In order for this to work you must override the connector properties, which means you must set up the connector properties for this transaction, using one of the load methods (such as LoadConnectorPropFromNode, LoadConnectorPropFromTrx, and so on.)

This property is read-write.

TransactionID

Description

This property returns the transaction ID as a string. This is used to uniquely identify a request.

This property is read-only.

UserName

Description

This property returns the user name associated with the message, as a string.

This property is read-only.

VisitedNodes

Description

This property returns an array of string containing the names of all the nodes visited by the message. This is useful when a message is being propagated across multiple nodes.

This property is read-only.

WSA_Action

Description

Use this property to specify the Web Services Addressing (WS-Addressing) action for the message. The WS-Addressing action is defined as a string.

This property is read-write.

WSA_FaultTo

Description

Use this property to specify the WS-Addressing fault end point for the message. The WS-Addressing fault end point is defined as a string.

If this property is not null, a message ID (the WSA_MessageID property) must also be defined.

This property is read-write.

WSA_MessageID

Description

Use this property to specify the WS-Addressing message ID for the message. The WS-Addressing message ID is defined as a string.

This property is read-write.

WSA_ReplyTo

Description

Use this property to specify the WS-Addressing reply-to address for the message. The WS-Addressing reply-to address is defined as a string.

This property is read-write.

WSA_To

Description

Use this property to specify the WS-Addressing destination for the message. The WS-Addressing destination is defined as a string.

This property is read-write.

IBConnectorInfo Collection

A IBConnectorInfo collection object is returned from the IBConnectorInfo IBInfo class property.

See Also

Chapter 25, "Message Classes," IBConnectorInfo, page 1366

IBConnectorInfo Collection Methods

In this section, we discuss the IBConnectorInfo collection methods. The methods are discussed in alphabetical order.

AddConnectorProperties

Syntax

`AddConnectorProperties``(`*`Name`*`,`*`Value`*`,`*`Type`*`)`

Description

Use the AddConnectorProperties method to add a set of connector properties to a connector.

Parameters

Parameter	*Description*
Name	Specify the name of the property as a string.
Value	Specify the value of the property as a string.

Parameter	Description
Type	Specify the type of the property as a string. The valid values are:

Constant Value	Description
%Property	Add a property type property.
%Header	Add a header type property.

Returns

A Boolean value: true if the connector properties are added successfully, false otherwise.

Example

The following are examples of typical name/value pairs.

```
&b1 = &MSG.IBInfo.IBConnectorInfo.AddConnectorProperties("URL", "http:⇒
//finance.yahoo.com/d/quotes.txt/?symbols=PSFT&format=l1c1d1t1", %Property);

&b2 = &MSG.IBInfo.IBConnectorInfo.AddConnectorProperties("sendUncompressed", ⇒
"Y", %Header);

&b3 = &MSG.IBInfo.IBConnectorInfo.AddConnectorProperties("FilePath", ⇒
"C:\Temp", %Property);
```

The following example demonstrates setting a passive connection for the FTP target connector:

```
&b4 = &MSG.IBInfo.IBConnectorInfo.AddConnectorProperties ("FTPMODE", ⇒
"PASSIVE", %Property);
```

See Also

Chapter 25, "Message Classes," DeleteConnectorProperties, page 1377 and Chapter 25, "Message Classes," ClearConnectorProperties, page 1376

AddQueryStringArg

Syntax

AddQueryStringArg*(Name,Value)*

Description

Use the AddQueryStringArg method to add query string arguments to the outbound request. The query string arguments are used by the HTTP connector to step parameters in the URL.

Parameters

Parameter	*Description*
Name	Specify the name of the query string argument as a string.
Value	Specify the value for the query string argument as a string.

Returns

A Boolean value: true if the query string argument is added successfully, false otherwise.

See Also

Chapter 25, "Message Classes," ClearQueryStringArgs, page 1377; Chapter 25, "Message Classes," DeleteQueryStringArg, page 1378; Chapter 25, "Message Classes," GetNumberOfQueryStringArgs, page 1381; Chapter 25, "Message Classes," GetQueryStringArgName, page 1381 and Chapter 25, "Message Classes," GetQueryStringArgValue, page 1382

ClearConnectorProperties

Syntax

```
ClearConnectorProperties()
```

Description

Use the ClearConnectorProperties method to clear all the properties in a connector before setting them.

Parameters

None.

Returns

None.

See Also

Chapter 25, "Message Classes," AddConnectorProperties, page 1374 and Chapter 25, "Message Classes," DeleteConnectorProperties, page 1377

ClearQueryStringArgs

Syntax

`ClearQueryStringArgs``()`

Description

Use the ClearQueryStringArgs method to clear all of the existing query string arguments.

Use the DeleteQueryStringArg method if you want to remove a specific query string argument.

Parameters

None.

Returns

None.

See Also

Chapter 25, "Message Classes," DeleteQueryStringArg, page 1378

DeleteConnectorProperties

Syntax

`DeleteConectorProperties`*`(Name)`*

Description

Use the DeleteConnectorProperties to delete a specific connector property.

Use the ClearConnectorProperties method to remove all the properties.

Parameters

Parameter	***Description***
Name	Specify the name of the connector property you want to delete.

Returns

A Boolean value: true if the property is deleted successfully, false otherwise.

See Also

Chapter 25, "Message Classes," ClearConnectorProperties, page 1376

DeleteQueryStringArg

Syntax

`DeleteQueryStringArg`*`(Name)`*

Description

Use the DeleteQueryStringArg method to delete a specific query string argument.

Use the ClearQueryStringArg method to delete all of the query string arguments.

Parameters

Parameter	***Description***
Name	Specify the name of the query string argument that you want to delete.

Returns

A Boolean value: true if the query string argument is deleted successfully, false otherwise.

See Also

Chapter 25, "Message Classes," ClearQueryStringArgs, page 1377

GetConnectorPropertiesName

Syntax

`GetConnectorPropertiesName`*`(Index)`*

Description

Use the GetConnectorPropertiesName to return the name of the connector property in the numeric position specified by *Index.*

Parameters

Parameter	*Description*
Index	Specify the numeric position of the connector property name that you want to access.

Returns

A string containing the name of a connector property.

Example

```
For &I = 1 to &Msg.IBInfo.IBConnectorInfo.GetNumberOfConnectorProperties();
  &PropName = &Msg.IBInfo.IBConnectorInfo.GetConnectorPropertiesName(&I)
  /* do processing */
End-For;
```

GetConnectorPropertiesType

Syntax

GetConnectorPropertiesType*(Index)*

Description

Use the GetConnectorPropertiesType method to return the type of the connector property specified by its numeric position by *Index.*

Parameters

Parameter	*Description*
Index	Specify the numeric position of the connector property type that you want to access.

Returns

A string containing the type of the specified connector.

GetConnectorPropertiesValue

Syntax

`GetConnectorPropertiesValue`*`(Index)`*

Description

Use the GetConnectorPropertiesValue method to return the value of the connector property specified by its numeric position by *Index.*

Parameters

Parameter	***Description***
Index	Specify the numeric position of the connector property type that you want to access.

Returns

A string containing the value of the specified connector.

GetNumberOfConnectorProperties

Syntax

`GetNumberOfConnectorProperties``()`

Description

Use the GetNumberOfConnectorProperties method to determine the number of connector properties.

Parameters

None.

Returns

A number.

GetNumberOfQueryStringArgs

Syntax

`GetNumberOfQueryStringArgs``()`

Description

Use the GetNumberOfQueryStringArgs method to determine the number of query string arguments.

Parameters

None.

Returns

A number.

GetQueryStringArgName

Syntax

`GetQueryStringArgName`*`(Index)`*

Description

Use the GetQueryStringArgName method to access the name of the query string argument by its numeric position as specified by *Index.*

Parameters

Parameter	***Description***
Index	Specify the numeric position of the query string argument name that you want to access.

Returns

A string containing the name of a query string argument.

GetQueryStringArgValue

Syntax

`GetQueryStringArgValue`*`(Index)`*

Description

Use the GetQueryStringArgValue method to access the value of the query string argument by its numeric posistion as specified by *Index*.

Parameters

Parameter	***Description***
Index	Specify the numeric position of the query string argument value that you want to access.

Returns

A string containing the value of a query string argument.

IBConnectorInfo Collection Properties

In this section, we discuss the IBConnectorInfo collection properties. The properties are discussed in alphabetical order.

ConnectorClassName

Description

Use this property to identify the name of the target connector to invoke as a string.

This property is read-write.

ConnectorName

Description

Use this property to identify the target connector to invoke to send to the message, as a string.

This property is read-write.

Cookies

Description

Use this property to access the cookies associated with a message.

You can accept a synchronous response message containing cookies, save those cookies in a global variable, and later return them to the target node in an outbound synchronous or asynchronous request message.

You can access this property only in an inbound synchronous response message or an outbound request message.

This property is read-write.

Example

The following example retains the cookies from a response message to a global variable:

```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;

&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* Send the synchronous request; the return value is the response
message object */
&SalesResponse = &SalesRequest.SyncRequest();

/* Retrieve cookies from the response message */
&SalesCookies = &SalesResponse.IBInfo.IBConnectorInfo.Cookies;
```

The following example retrieves the previously retained cookies from the global variable and inserts them into a new request message:


```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;

&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(Message.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* Insert the cookies in the request message */
&SalesRequest.IBInfo.IBConnectorInfo.Cookies = &SalesCookies;

/* Send the asynchronous request */
%IntBroker.Publish(&SalesRequest);
```

PathInfo

Description

This property is specific to incoming HTTP requests. This is the path information extracted from the request, represented as a string.

This property is read-write.

RemoteFrameworkURL

Description

Use this property to identify the URL to which to send a message, as a string. This value overrides the server URL specified in the Project Definitions section.

This property is read-write.

Chapter 26

Mobile Classes

This chapter provides an overview of the Mobile classes and discusses the following topics:

- Mobile classes overview.
- Mobile terms.
- Mobile hierarchy.
- HTML areas used in mobile.
- Debugging in mobile.
- Error handling.
- Data types for mobile.
- Scope of a mobile object.
- Mobile classes reference.

Important! PeopleSoft Mobile Agent is a deprecated product. The mobile classes currently exist for backward compatibility only.

See Also

Chapter 41, "SyncServer Class," page 2261

Mobile Classes Overview

PeopleSoft Mobile Agent extends the functionality of PeopleSoft Pure Internet Architecture to disconnected mobile devices, allowing users to continue working with their PeopleSoft applications on a laptop computer or personal digital assistant (PDA) while disconnected from the Internet or a local network.

When you create mobile pages for display on a mobile device (a laptop or PDA), you create the following in Application Designer:

- Synchronizable Component Interfaces.
- Mobile pages based on your synchronizable Component Interfaces.

In addition, you may create PeopleCode programs that manipulate the data. You use the mobile classes in a mobile application to work with the data on a mobile device.

This document provides an overview of using the mobile classes, including:

- Understanding the Mobile Device and the Component Interface
- Understanding differences between Mobile and PIA

See Also

Chapter 41, "SyncServer Class," page 2261

Understanding the Mobile Device and the Component Interface

When the synchronization server and the mobile device are synchronized, the following items are downloaded to the mobile device:

- instances of synchronizable Component Interfaces.
- mobile pages.
- associated mobile PeopleCode programs.

When a user of Mobile Agent synchronizes with the synchronization server the first time, the relevant data for the synchronizable Component Interface is loaded to the mobile device.

During subsequent synchronizations, the data on the mobile device is synchronized with the database on the synchronization server.

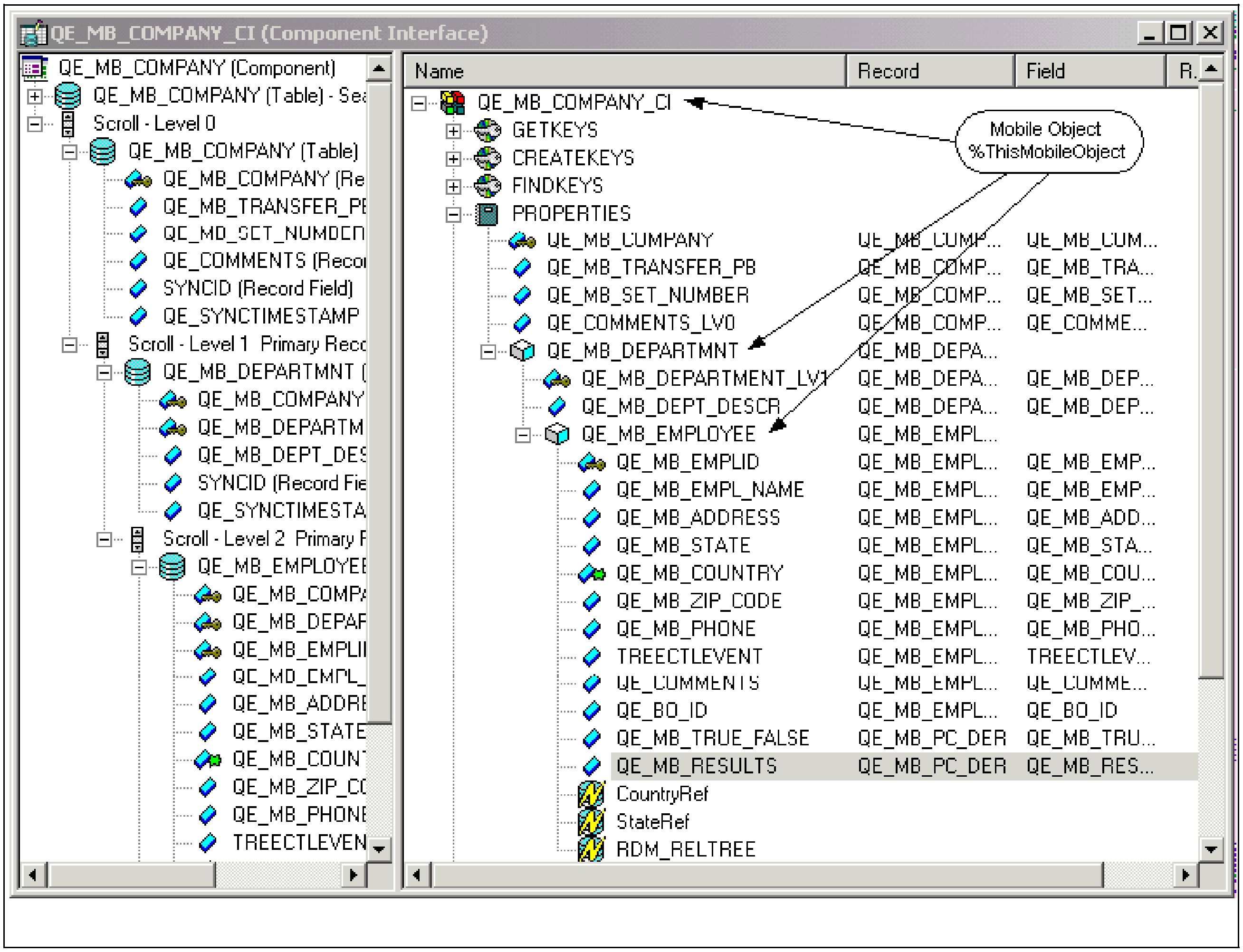
This section discusses the items that are downloaded to the mobile device. For more information about the synchronization process, see the Synchronization Server documentation.

See Chapter 41, "SyncServer Class," page 2261.

Synchronizable Component Interface

A synchronizable Component Interface is a Component Interface that is created specifically for use with a mobile device. This means that the Component Interface definition has been specified as synchronizable on the Component Interface properties page.

The following shows a synchronizable Component Interface.



Synchronizable Component Interface

Level zero of the synchronizable component interface, as well as the data in the collections, are known as *mobile objects*, and are described in the following sections.

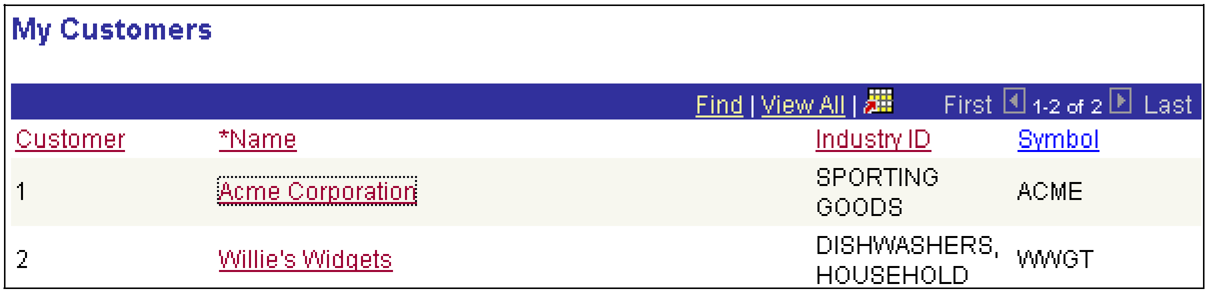
Mobile Page

The Mobile Page enables a mobile device user to change data in the synchronizable Component Interface that underlies the Mobile Page. The changes to the Component Interface data are sent back to the PeopleSoft database when the mobile device is synchronized.

Mobile pages have *list views* and *detail views.*

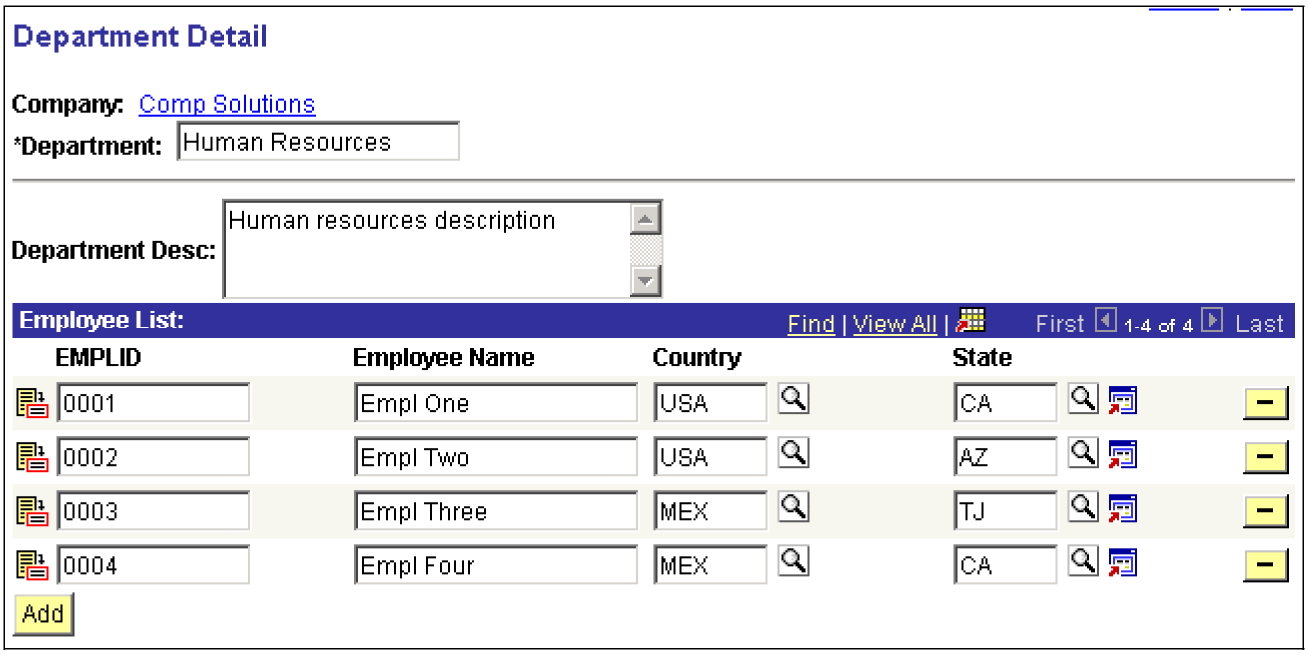
- List view pages are created from the top level of a component interface (the level zero), or a collection in a component interface. A list view can also be thought of as a search page.
- Detail view pages are created from the content properties (fields) in a mobile page. Where and how a property displays on a page is determined by the content properties page for that property (field).

The following is the list view for a mobile page, generated by the level zero of the synchronizable component interface.



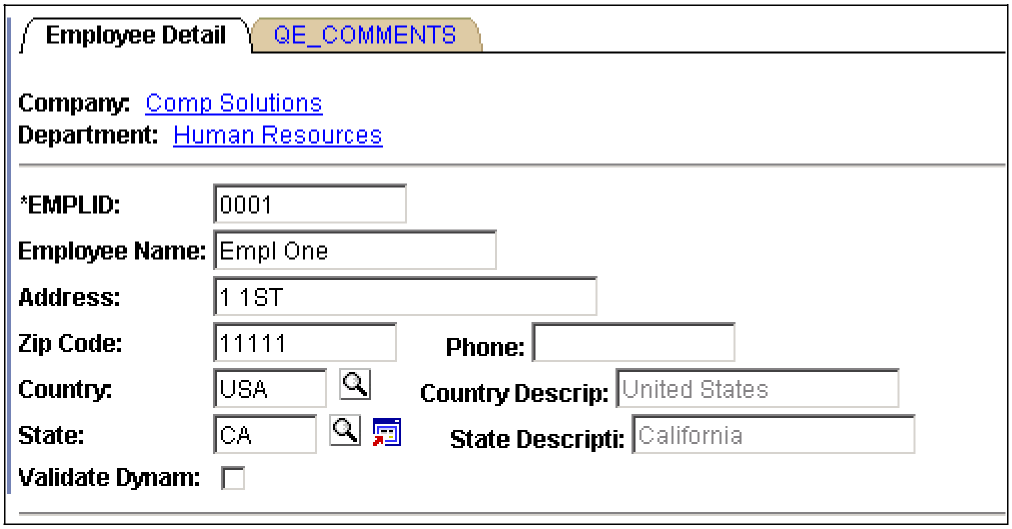
Mobile Page: level zero List View

The following is a list view generated by a collection in a synchronizable component interface.



Mobile page: Collection List View

The following is the detail view generated by the properties (fields) of the synchronizable component interface.



Mobile page: Detail View

Associated Mobile PeopleCode Programs

Access PeopleCode associated with a Component Interface by right clicking on an open synchronizable Component Interface in Application Designer and selecting ViewPeopleCode.

The following is a PeopleCode program associated with the OnChange Mobile Component Interface event on a synchronizable Component Interface. The OnChange event is associated with a property on the page. The PeopleCode program uses a *mobile object* to access the QE_MB_EMPLID property on level zero on the synchronizable Component Interface that has been downloaded to the mobile device. The mobile object represents the PIA equivalent of a row of data, and can be used at any level in the data hierarchy, level zero, level one, level two, and so on. It is what is displayed in the Detail View of a Mobile Page.

When a mobile event occurs on a mobile object, such as a mobile object property being changed (the OnChange event) or a mobile object being loaded into the working set (the OnInit event), the mobile object is then available in the event's PeopleCode program using the %ThisMobileObject system variable.

```
If %ThisMobileObject.IsNew() Then

/* do further check before save */

End-if;
```

Considerations Using the Mobile Object

A mobile object is the means in PeopleCode to access the synchronizable Component Interface. Mobile objects exist in relationship to each other in a hierarchy, which has a Component Buffer-like structure. A mobile object is just one row. A Data Collection is a collection of rows, each row being a mobile object.

The mobile object hierarchy has the equivalent of rowsets as data collections, rows as mobile objects, records and fields as properties. The topmost mobile object for a Component Interface is the equivalent of level zero on the underlying Component Interface. A Component Interface can also contain levels, which map to mobile object data collections. (These levels, in PIA, map to scrolls on a page, or in the Component Buffer, to rowsets.)

In the example in this section, the hierarchy of the mobile objects is:

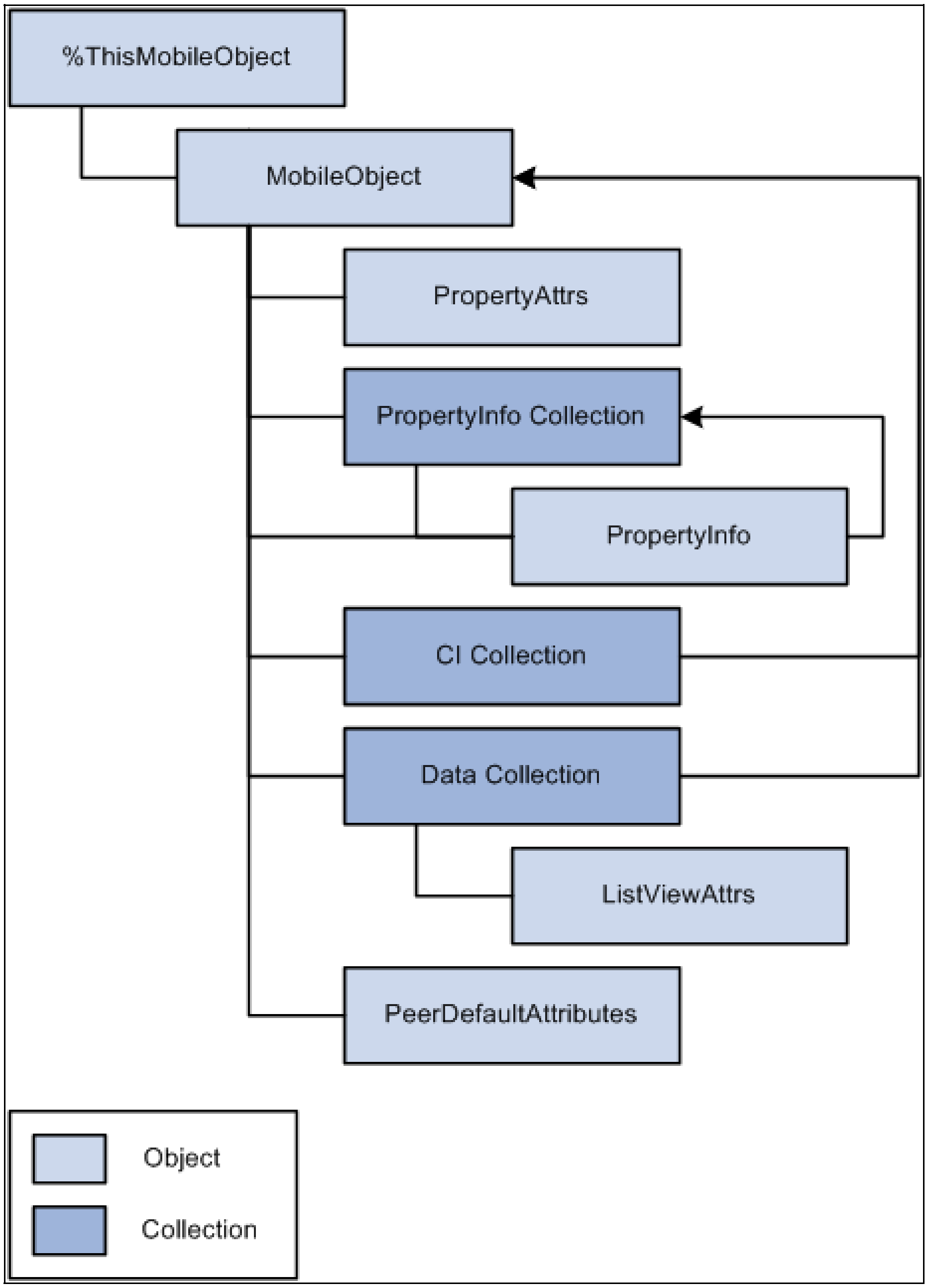

```
QE_MB_COMPANY
    QE_MB_DEPARTMNT
        QE_MB_EMPLOYEE
```

- QE_MB_COMPANY — level zero.
- QE_MB_DEPARTMNT — level one. This is a collection of mobile data objects.
- QE_MB_EMPLOYEE — level two. This is a collection of mobile data objects.

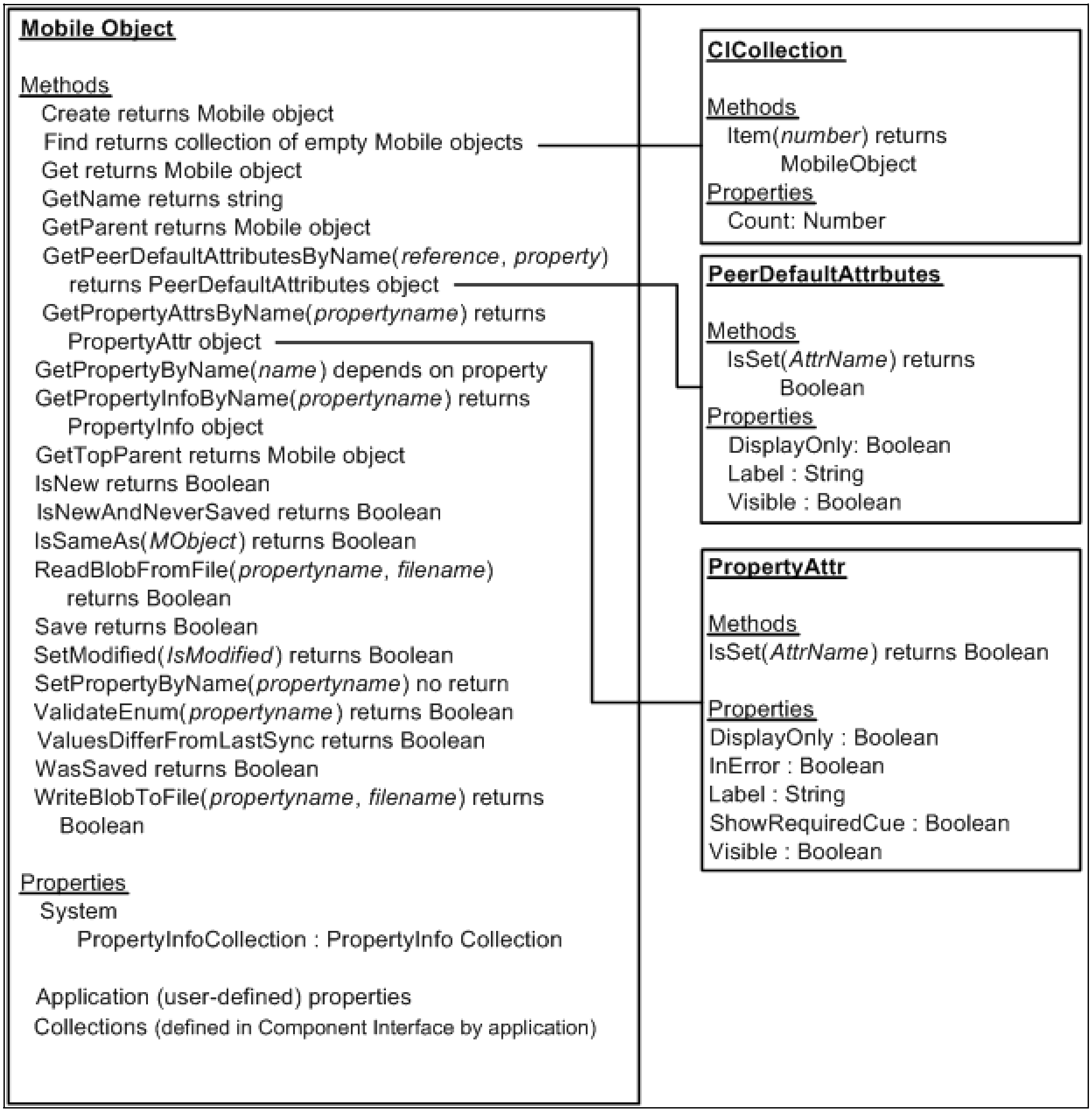
Mobile Classes Hierarchy

Mobile objects exist in relationship to each other in a hierarchy. The following diagram shows the different classes, and how one class can be returned from a method or a property in another class.

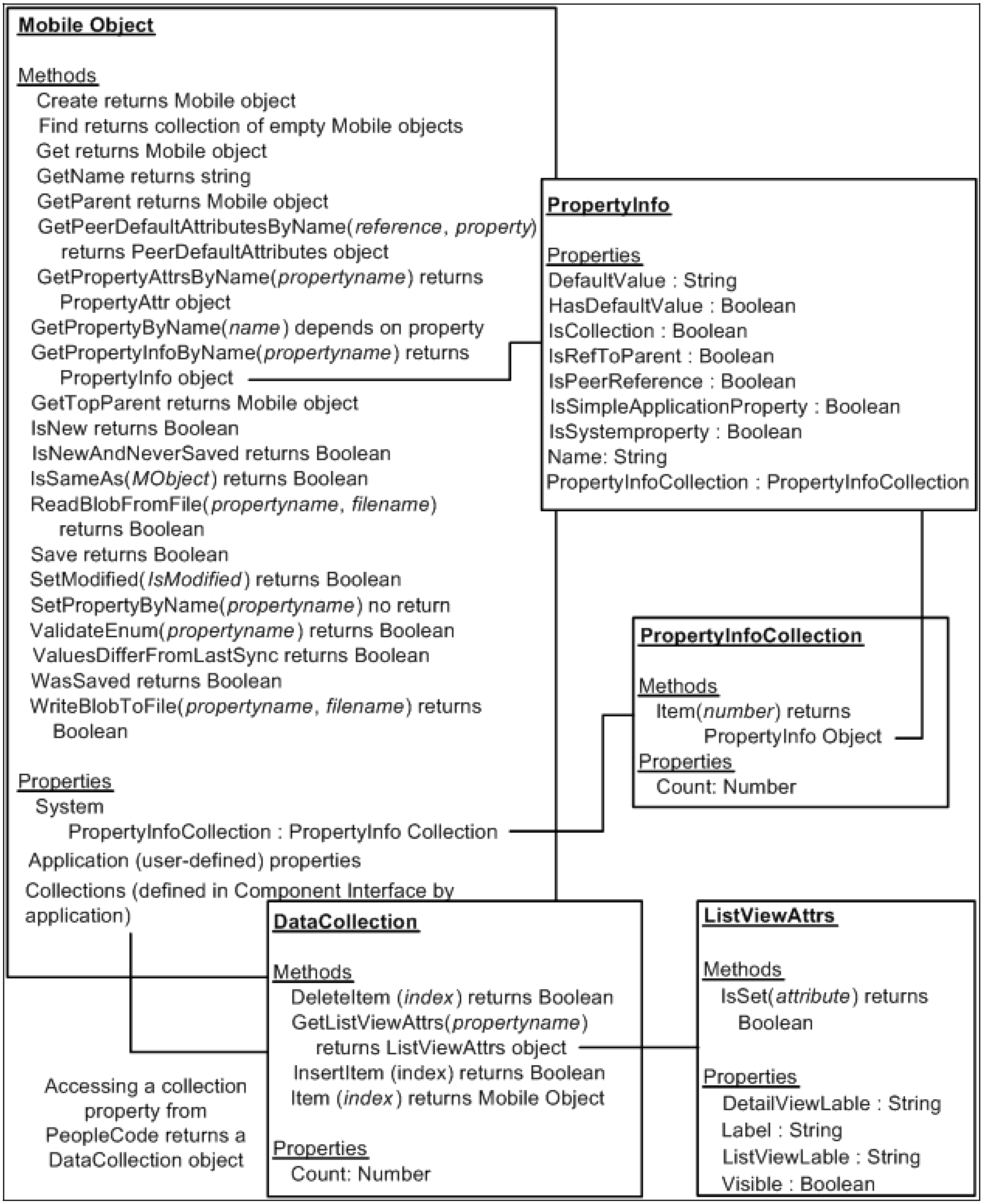
The following presents the mobile object hierarchy in a class diagram:



Mobile classes hierarchy



Mobile classes methods (part 1 of 2)



Mobile classes methods (part 2 of 2)

The following is a list of the various classes, with a brief description of each.

Mobile Classes	***Description***
Mobile object	Represents the PIA equivalent of a row of data, and can be used at any level in the data hierarchy, level zero, level one, level two, and so on. It is what is displayed in the Detail View of a mobile page.
CI collection	Contains a collection of empty Mobile objects.
Data collection	Used to access a level on a mobile page, such as level one, level two, and so on.
ListViewAttrs	Used to access the display attributes for a list view.
PeerDefaultAttributes	Used to access the default display attributes for a peer reference.
PropertyAttrs	Used to access the display attributes for a property.
PropertyInfo	Used to access the attributes of a property (field) on a mobile page.
PropertyInfoCollection	Contains a collection of PropertyInfo objects. Used when a property on a mobile page is actually a collection (such as a reference to level one, level two, and so on.)

Debugging in Mobile

The psmobile.ini file contains debug settings for mobile PeopleCode.

For a laptop, you can edit the psmobile.ini file on the mobile device. This file is either in the C:\Windows or C:\WinNT directory.

For a PDA, you must copy the psmobile.ini file to the laptop or desktop using ActiveSync, edit it there, then copy it back.

Use the DebugEnabled value in the psmobile.ini file to control whether debugging is enabled for your application. If you specify a 1 for this value, you can use the PeopleCode debugger with your mobile application for debugging.

In addition, you can use the TracePC value in the psmobile.ini file to specify that a trace is to be run, as well as what type of operations in the application should be traced.

The following is an example of the psmobile.ini file.


```
[PSMOBILE]
Language=ENG
Port=8080
HomeDirectory=C:\Program Files\PeopleSoft\PeopleSoft Mobile Agent
MaxCachedStatements=100
MaxCachedObjects=100
debug=1
[SYNC]
SyncGateway=http://fsampson040502.corp.peoplesoft.com:80/SyncServer
SyncLogVerbose=0
[PCDEBUG]
DebugEnabled=0
DebugDatabase=Your Database Name
DebugOprid=Your OPRID
DebugIPAddress=nnn.nnn.nnn.nnn
DebugPort=9500
[Trace]
;-------------------------------------------------------------
; PeopleCode Tracing Bitfield
;
; Bit       Type of tracing
; ---       ---------------
; 1         - Trace instructions
; 2         - List the program
; 4         - Show assignments to variables
; 8         - Show fetched values
; 16        - Show stack
; 64        - Trace start of programs
; 128       - Trace external function calls
; 256       - Trace internal function calls
; 512       - Show parameter values
; 1024      - Show function return value
; 2048      - Trace each statement in program
; Dynamic change allowed for TracePC and TracePCMask
TracePC=0
```

Debug Settings

The following are the debug settings you can set in the psmobile.ini file.

DebugEnabled	Specify whether debugging is enabled or not. Use a 0 to turn off debug mode, a 1 to turn on debug mode.
DebugDatabase	Specify the actual database (not the domain) that application server uses with the PeopleTools PeopleCode debugger (this should be the database that is used for synchronizing).
DebugOprid	Specify the userId you use to log onto the PeopleTools PeopleCode debugger.
DebugIPAddress	Specify the IP address or machine name of the application server.

DebugPort	Specify the port number that the application server (PSDBGSRV) has been configured to use.

Trace Settings

Use the TracePC value in the psmobile.ini file to specify that a trace is to be run, as well as what type of operations in the application should be traced.

All trace information is written to a file named `psmobile.log`, located in the directory defined by `%temp%`.

Note. Be aware that tracing logs can grow to a large size. Only specify tracing when necessary, and only trace critical operations.

The following are the trace settings you can specify using TracePC. You can specify more than one trace by adding the numbers together.

0	Turn all tracing off.
1	Trace all instructions.
2	List the program being traced.
4	Show all assignments to all variables.
8	Show all fetched values.
16	Show stack.
64	Trace start of programs.
128	Trace external function calls.
256	Trace internal function calls.
512	Show all parameter values.
1024	Show function return value.
2048	Trace each statement in program.

Error Handling

For initial error handling, check the return values of methods and functions as they are used. When you want to check for errors depends on your application. However, if you check for errors after every assignment, you may see a performance degradation.

Mobile does not use collections of errors or messages (PSMESSAGES), as is the case with PeopleSoft Pure Internet Architecture. However, you can specify a message set and number for use with a mobile page and with mobile properties. This is done in Application Designer, to ensure that the correct messages get downloaded to the mobile device.

Once the messages have been distributed to the mobile device, you can access them using PeopleCode the same way you would access any message in the message catalog, using the following functions:

- MessageBox
- MsgGet
- MsgGetText
- MsgGetExplainText

In the following example, the program checks for an invalid value for a field, and issues a message if the value is incorrect. This code would be in an OnChangeEdit event associated with a property.

```
&Value = %ThisMobileObject.EmailID;

If Not(Find("@", &Value)) Then
     /* invalid email address probably */
     MessageBox(0, "", 525, 2, "Message Not Found");

     &PropAttrs = %ThisMobileProperty.GetPropertyAttrsByName("EmailID");
     &PropAttrs.InError = True

End-If;
```

Data Types for Mobile

All mobile objects are declared as type ApiObject. For example:

```
Local ApiObject &MyMobileCollection;
```

```
Local ApiObject &MyMobileObject = %ThisMobileObject;
```

Note. Mobile objects can be declared only as Local.

Scope of Mobile Objects

A mobile object can be instantiated from PeopleCode. In addition, it can only be instantiated within the mobile events.

Variables in Mobile PeopleCode

Local variables are the only type of PeopleCode variables supported in mobile.

Mobile Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateMobileTree

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," IsDisconnectedClient

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," MSFGetNextNumber

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TransferMobilePage

Session Class Method

The following is the Session class method used with Mobile.

See Also

Chapter 38, "Session Class," page 2179

GetCompIntfc

Syntax

`GetCompIntfc``([`**`COMPINTFC.`**`]`*`ComponentName`*`)`

Description

The GetCompIntfc method returns a reference to an empty mobile object. Mobile objects are the means of accessing the synchronizable Component Interface in PeopleCode, so this is similar to creating an empty Component Interface in PIA.*ComponentName*, when used by itself, takes a string value. If you use COMPINTFC.*ComponentName* it is not a string value; it is a constant that automatically is renamed in your code if the Component Interface definition is ever renamed.

Parameters

Parameter	***Description***
ComponentName	Specify the name of the synchronizable Component Interface. You can either use *ComponentName* by itself or prefaced with the keyword **COMPINTFC.**

Returns

This method returns an empty mobile object if successful, or Null on error (such as *ComponentName* specifies a nonexistent synchronizable Component Interface).

Example

```
Local ApiObject &SaveCustomer, &Customer;

&SaveCustomer = %ThisMobileObject.MyFavoriteCustomer;

&Customer = %Session.GetCompIntfc(COMPINTFC.CUSTOMER);
&Customer.Name="Acme Applicances";

if (&Customer.Get() <> true) then;
   if (&Customer.Create() <> true) then;
      ErrorHandler();
   Else
      &Customer.Save();
   End-If;
End-if;

%ThisMobileObject.MyFavoriteCustomer = &Customer;
```

Mobile Class

The Mobile object represents the PIA equivalent of a row of data. It is what is displayed in the detail view of a mobile page.

The mobile object is accessed using the %ThisMobileObject system variable.

Mobile Class Methods

The following are the Mobile class methods, in alphabetical order.

Note. With the Mobile class methods, and unlike the Component Interface class methods, all properties that have been set are used in the search. This provides more flexibility, and does not require setting up specific Get, Create, or Find keys.

Create

Syntax

```
Create()
```

Description

The Create method associates the mobile object with a new, open mobile object. It is available only on an empty mobile object that is returned by the GetCompIntfc session method.

Parameters

None.

Returns

A Boolean value: True if object was successfully created, False otherwise.

Example

```
Local ApiObject &SaveCustomer, &Customer;

&SaveCustomer = %ThisMobileObject.MyFavoriteCustomer;

&Customer = %Session.GetCompIntfc(COMPINTFC.CUSTOMER);
&Customer.Name="Acme Applicances";

if (&Customer.Get() <> true) then;
   if (&Customer.Create() <> true) then;
     ErrorHandler();
   Else
     &Customer.Save();
   End-If;
End-if;

%ThisMobileObject.MyFavoriteCustomer = &Customer;
```

See Also

Chapter 26, "Mobile Classes," GetCompIntfc, page 1398

Find

Syntax

Find()

Description

The Find method returns a reference to a CI collection that is populated with a collection of empty mobile objects. The properties that the application developer sets within the empty mobile object are used as the Find keys. This method is available only on an empty mobile object that is returned by the GetCompIntfc session method. Using it on a non-empty mobile object causes an error.

This collection may have zero elements in it if no items matching the key values you set were found. If you set no keys, all of the mobile objects of the type specified by GetCompIntfc are returned.

While the PIA Component Interface Find method limits the number of returned instances to 300, the mobile Find method has no limit.

The Find method returns only the topmost parent mobile objects. You do not have to set values for all the key values. After you have a collection, you can use the CI Collection method Item to return the empty mobile object, and then use Get on the mobile object to instantiate it.

Note. Wildcards are not implemented for this release.

Parameters

None.

Returns

A CICollection collection of mobile objects. You must use the Get method on these mobile objects before you can use them.

Example

```
/* Error checking left out for clarity */

&MyCI = %Session.GetCompIntfc(CompIntfc.QE_MB_COMPANY_CI);
&MyCI.QE_MB_COMPANY = "Acme Appliances";
&MyCIFindCollection = &MyCI.Find();
&Acme = &MyCIFindCollection.Item(1);
&Acme.Get();
```

Get

Syntax

```
Get()
```


Description

The Get method associates a mobile object with the user-defined properties that have been explicitly set on the object.

Parameters

None.

Returns

A Boolean value: True if mobile object was successfully retrieved, False otherwise.

Example

```
Local ApiObject &SaveCustomer, &Customer;

&SaveCustomer = %ThisMobileObject.MyFavoriteCustomer;

&Customer = %Session.GetCompIntfc(COMPINTFC.CUSTOMER);
&Customer.Name="Acme Applicances";

if (&Customer.Get() <> true) then;
    if (&Customer.Create() <> true) then;
      ErrorHandler();
  Else
      &Customer.Save();
  End-If;
End-if;

%ThisMobileObject.MyFavoriteCustomer = &Customer;
```

GetName

Syntax

GetName()

Description

The GetName method returns the class name of the mobile object.

Parameters

None.

Returns

The class name of the mobile object as a string. This is the name of the synchronizable Component Interface, or one of the levels.

Example

```
&GetName = &qe_mb_obj_setup.GetName();

/*   Write the name of the Object to the Results field */
&qe_mb_obj.QE_MB_RESULTS = "The name of this Mobile Object is: " | &GetName;
```

GetParent

Syntax

GetParent()

Description

The GetParent method returns a mobile object that is the parent of the Mobile object that is executing the method. If you are at the top level (level zero), it returns itself.

Parameters

None.

Returns

The parent mobile object.

Example

```
/* This loop will continue getting parent level, until level 0 is reached.  */
&obj = %ThisMobileObject;
Repeat
    &obj = &obj.getparent();
    /* Do some processing */
until (&obj.IsSameAs(%ThisMobileObject.gettopparent()));
```


GetPeerDefaultAttributesByName

Syntax

`GetPeerDefaultAttributesByName`*`(Reference,Property`*`)`

Description

When a peer reference does not have a value associated with it, the display attributes can only be accessed using the PeerDefaultAttributes class. After a peer reference has values, it can be accessed using the ListViewAttrs class.

Use the GetPeerDefaultAttributesByName method to access the default peer reference attributes for the specified property on the peer reference.

Parameters

Parameter	***Description***
Reference	Specify the name of a peer reference, as a string.
Property	Specify a property on the peer reference, as a string.

Returns

A reference to a PeerDefaultAttributes class.

Example

```
%ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB", "ADDRESS").⇒
Visible = True;
```

See Also

Chapter 26, "Mobile Classes," PeerDefaultAttributes Class, page 1439

GetPropertyAttrsByName

Syntax

`GetPropertyAttrsByName`*`(propertyname)`*

Description

The GetPropertyAttrsByName method returns a reference to a PropertyAttrs object, used to modify display attributes of properties.

Parameters

Parameter	*Description*
propertyname	The name of the property.

None.

Returns

A PropertyAttrs object, used to modify display attributes of properties.

Example

```
%ThisMobileObject.GetPropertyAttrsByName("QE_MB_SET_DATE").Visible = False;
```

See Also

Chapter 26, "Mobile Classes," PropertyAttrs Class, page 1434

GetPropertyByName

Syntax

GetPropertyByName*(Name)*

Description

The GetPropertyByName method returns the value of a property. Generally this function is used only in applications that cannot get the names of the mobile object user-defined properties until runtime.

Parameters

Parameter	*Description*
Name	Specify the name of the property you want to access. The PeopleCode program terminates if this property does not exist.

Returns

The value of the property specified by *Name*. For a collection, it returns a reference to the collection.

Example

```
&GetPropertyByName_Char = &qe_mb_obj_setup.GetPropertyByName(&CharacterProperty⇒
Name);
/*Write the results of the GetPropertyByName method to the results Field   */

&qe_mb_obj.QE_MB_RESULTS = "The value of the Character Field: "
| &GetPropertyByName_Char | "
```

GetPropertyInfoByName

Syntax

GetPropertyInfoByName*(propertyname)*

Description

The GetPropertyInfoByName method returns a reference to a PropertyInfo object.

Parameters

Parameter	*Description*
propertyname	The name of the property.

Returns

A PropertyInfo object.

Example

```
&pinfo = %ThisMobileObject.GetPropertyInfoByName("QE_MB_SET_DATE");
if &pinfo.HasDefaultValue = True then
   %ThisMobileObject.QE_MB_SET_DATE = &pinfo.DefaultValue;
end-if;
```

See Also

Chapter 26, "Mobile Classes," PropertyInfo Class, page 1419

GetTopParent

Syntax

GetTopParent()

Description

The GetTopParent method returns a mobile object that is the topmost parent of the Mobile object.

Parameters

None.

Returns

A reference to the topmost parent mobile object as a mobile object. If the current mobile object is the topmost object, this method returns a reference to the current mobile object.

Example

```
/* Check if the created Mobile Object is at Level 0  */
If (&qe_mb_obj_setup.IsSameAs(&qe_mb_obj_setup.GetTopParent())) Then

   /* Set the Parent Level variable to 0.
   Results for this get set below the looping statement  */
   &Parent_Level = 0;

End-If;
```

IsNew

Syntax

IsNew()

Description

The IsNew method indicates if this object has never been synchronized.

Parameters

None.

Returns

A Boolean value: True if this object was created on the device and never synchronized, False if this object has been synchronized.

Example

```
&IsNew = &qe_mb_obj_setup.IsNew();

/*Write the results of the IsNew to the results Field   */

&qe_mb_obj.QE_MB_RESULTS = "The following is " | &IsNew | ": The Mobile Object is⇒
 new, and has never been synced.";
```

IsNewAndNeverSaved

Syntax

IsNewAndNeverSaved()

Description

The IsNewAndNeverSaved method indicates if this object was created on the device and has not yet been saved.

Parameters

None.

Returns

A Boolean value: True if this object was created on the device and has not yet been saved, False otherwise.

Example

```
&IsNewAndNeverSaved = &qe_mb_obj_setup.IsNewAndNeverSaved();

/*Write the results of the IsNewAndNeverSaved method to the results Field   */

&qe_mb_obj.QE_MB_RESULTS = "The following is " | &IsNewAndNeverSaved | ": This⇒
 Mobile Object is new, and has never been saved. "
```


IsSameAs

Syntax

IsSameAs*(MObject)*

Description

The IsSameAs method determines if one mobile object is the same as another. This is useful when you are trying to determine if the %ThisMobileObject is the topmost mobile object.

Parameters

Parameter	***Description***
MObject	Specify the mobile object to which this mobile object is being compared.

Returns

A Boolean value: True if this mobile object is the same as the mobile object, False otherwise.

Example

```
/*Check if the created Mobile Object is at Level 0  */
If (&qe_mb_obj_setup.IsSameAs(&qe_mb_obj_setup.GetTopParent())) Then

/* Set the Parent Level variable to 0. Results for this get set below
the looping statement  */
&Parent_Level = 0;

EndIf;
```

ReadBlobFromFile

Syntax

ReadBlobFromFile*(blobpropertyname,filename)*

Description

The ReadBlobFromFile method reads a file on your mobile device and writes it into the attachment property. The attachment property is a reference property on the synchronizable Component Interface; its purpose is to contain attachment data.

When the mobile user synchronizes from the mobile page, the attachment file data contained in the property is sent to the synchronization server. The size of the attachment on the synchronization server is updated to reflect the new contents.

If you use this method, your synchronized Component Interface must use an attachment: the synchronization server must use its attachment events, and must run the attachment method PutAttachment.

See Chapter 41, "SyncServer Class," Using Attachment Events, page 2269.

Working With Relative Paths

You can specify either an absolute or a relative path for *filename* .

If you specify a relative path, the path that is appended is the working directory for the PSMobile.exe file. The exact path depends on the installation of Mobile on the device.

For example, if you specified `\files\MyFile.ext` for *filename,* the actual path might be something like `c:\Program Files\PeopleSoft\PeopleSoft Mobile Agent\files\Myfile.ext`.

Parameters

Parameter	***Description***
blobpropertyname	A string specifying the name of the reference property used for the attachment.
FileName	This parameter is a string specifying the path and filename of the file that you use as an attachment.

Returns

This is a Boolean value: True if the attachment was read successfully; False otherwise.

Example

```
%ThisMobileObject.ReadBlobFromFile("Attachment", %ThisMobileObject.QE_MB_READ_⇒
WRITE | %ThisMobileObject.ATTACHUSERFILE);
```

Save

Syntax

Save()

Description

The Save method saves the entire mobile object hierarchy in which the mobile object is contained.

You do not generally call the Save method on %ThisMobileObject; that saving is done when the user clicks on the Save button.

Using this method does not save associated peer references; you must use the Save method on each peer references separately.

The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with OnSaveEdit, OnSavePreChange, and so on) are executed after you execute this method.

If you instantiate another mobile object with PeopleCode (using Get, Create) these objects are not automatically saved when the end user clicks the Save button. You must use the Save method with these instances separately.

Using the Save method on developer-instantiated mobile objects does not guarantee they will be saved. The save does not take effect until the end user clicks the Save button.

If you make changes to a developer-instantiated mobile object, the end user is not automatically prompted to save when trying to leave the page because it is not part of the current component.

Note. If a developer uses the Save method with a developer-instantiated component interface, and if the end user did not make any other changes to data in the current component interface, when the user leaves the current unit of work, they are *not* prompted to save their data.

To ensure that the data actually gets committed to the database, the developer can force a prompt to occur. This can be done using the following code:

```
%ThisMobileObject.SetModified(True);
```

Parameters

None.

Returns

A Boolean value: True if component was successfully saved, False otherwise.

Example

```
Local ApiObject &SaveCustomer, &Customer;
&SaveCustomer = %ThisMobileObject.MyFavoriteCustomer;
&Customer = %Session.GetCompIntfc(COMPINTFC.CUSTOMER);
&Customer.Name="Acme Applicances";
if (&Customer.Get() <> true) then;
    if (&Customer.Create() <> true) then;
      ErrorHandler();
  Else
      %ThisMobileObject.SetModified(True)
      &Customer.Save();
End-If;
End-if;
%ThisMobileObject.MyFavoriteCustomer = &Customer;
```


SetModified

Syntax

SetModified*(IsModified)*

Description

The SetModified method forces the mobile object to have the specified modified value of True or False.

Note. Not only does using this method change the overall modified state of the object, but when changing the modified state to False it also clears the modified flag on any properties that had been modified.

Parameters

Parameter	***Description***
IsModified	Specify whether this mobile object has been modified. This parameter takes a Boolean value: True, the mobile object has been modified, False, the object has not been modified.

Returns

The current setting of this mobile object *before* being modified by this method: True for modified before this setting, False for not modified before this setting.

Example

```
/*   Get the value of the TrueFalse check box, to see how to set
the modified setting.  */
If &qe_mb_obj.QE_MB_TRUE_FALSE = True Then
   &True_or_False = True;
   &SetModified = &qe_mb_obj_setup.SetModified(True);
Else
   &True_or_False = False;
   &SetModified = &qe_mb_obj_setup.SetModified( False);
End-If;
```

SetPropertyByName

Syntax

SetPropertyByName*(propertyname, value)*

Description

The SetPropertyByName method sets the value of a property. Generally this function is used only in applications that cannot set the names of the component interface properties until runtime.

Parameters

Parameter	***Description***
propertyname	The name of the property.
value	This is the value to which the property is set.

Returns

None. The PeopleCode program terminates if there is an error.

Example

```
%ThisMobileObject.SetPropertyByName("QE_MB_NUMBER", 77777);
```

ValidateEnum

Syntax

ValidateEnum*(propertyname)*

Description

The ValidateEnum method verifies if the value of a property is a valid enumeration value. Enumeration values are originally set in Application Designer, when the original records for the synchronizable Component Interface are designed.

Parameters

Parameter	***Description***
propertyname	Specify the name of the property as a string.

Returns

A Boolean value: True if the value is a valid dynamic enumeration, False otherwise.

Example

```
/* Validate if the value entered into Results field is a valid dynamic enumeration⇒
 */
&ob = %ThisMobileObject;

&Validate_DE = &ob.ValidateEnum(&ob.QE_MB_RESULTS);

/*Enter Results into Results Field  */
&ob.QE_MB_RESULTS = "The following statement is " | &Validate_DE | ": The Property⇒
 " | &ob.QE_MB_RESULTS | " is a valid Dynamic Enumeration.";
```

ValuesDifferFromLastSync

Syntax

ValuesDifferFromLastSync()

Description

The ValuesDifferFromLastSync method indicates if at least one property has a value that differs from its value since the last synchronization.

If this object has been created on the mobile device, that is, when there is no previous sync value, the first saved values are used as the comparison values for this property.

This is a true member by member comparison of data values, not modified flags. The SetModified method has no effect on this method.

Parameters

None.

Returns

A Boolean value: True if at least one property in this object has a value that differs from its original value, False otherwise.

Example

```
&ValuesDifferFromLastSync = &qe_mb_obj_setup.ValuesDifferFromLastSync();

/*Write the results of the ValuesDifferFromLast Sync method to the results Field*/

&qe_mb_obj.QE_MB_RESULTS = "The following is " | &ValuesDifferFromLastSync | ": At⇒
 least one property has a value that differs from the values last synced."
```

WasSaved

Syntax

WasSaved()

Description

The WasSaved method indicates if this object has been saved since it was created or last synchronized.

Parameters

None.

Returns

A Boolean value: True if this object has been saved since it was created or last synchronized, False otherwise.

Example

```
&WasSaved = &qe_mb_obj_setup.WasSaved();

/*Write the results of the WasSaved method to the results Field   */

&qe_mb_obj.QE_MB_RESULTS = "The following is " | &WasSaved | ": This Mobile Object⇒
 has been saved since it was last synced, or created. "
```

WriteBlobToFile

Syntax

WriteBlobToFile*(blobpropertyname,filename)*

Description

The WriteBlobToFile method reads the attachment property and writes it into a file on your mobile device. The attachment property is a reference property on the synchronizable Component Interface; its purpose is to contain attachment data.

When the mobile user synchronizes from the mobile page, this attachment file data contained in the property is received from the synchronization server.

If you use this method, your synchronized Component Interface must use an attachment: the synchronization server must use its attachment events, and must run the attachment method GetAttachment.

See Chapter 41, "SyncServer Class," Using Attachment Events, page 2269.

Working With Relative Paths

You can specify either an absolute or a relative path for *filename* .

If you specify a relative path, the path that is appended is the working directory for the PSMobile.exe file. The exact path depends on the installation of Mobile on the device.

For example, if you specified `\files\MyFile.ext` for *filename,* the actual path might be something like `c:\Program Files\PeopleSoft\PeopleSoft Mobile Agent\files\Myfile.ext`.

Parameters

Parameter	***Description***
blobpropertyname	A string specifying the name of the reference property that is used for the attachment.
FileName	A string specifying the path and filename of the attachment file. This parameter takes a string value.

Returns

This is a Boolean value: True if the attachment was written successfully; False otherwise.

Example

```
%ThisMobileObject.WriteBlobToFile("Attachment", %ThisMobileObject.QE_MB_READ_WRITE⇒
 | %ThisMobileObject.ATTACHUSERFILE);
```

Mobile Class Property

The following is the Mobile class property.

Note. Normally, properties for mobile objects are the equivalent of the user-define properties on the synchronizable Component Interface. The properties described here should not be used for user-defined properties when one designs a synchronizable Component Interface in the Application Designer.

PropertyInfoCollection

Description

The PropertyInfoCollection property returns a reference to a collection of PropertyInfo objects that describe the shape of the mobile object.

This property is read-only.

See Also

Chapter 26, "Mobile Classes," PropertyInfo Class, page 1419

PropertyInfoCollection

The PropertyInfoCollection objects contains a collection of PropertyInfo objects.

A PropertyInfoCollection is returned from the following:

- a mobile object
- a PropertyInfo object

See Also

Chapter 26, "Mobile Classes," PropertyInfoCollection, page 1417

Chapter 26, "Mobile Classes," PropertyInfoCollection, page 1423

PropertyInfoCollection Method

The following is the method for the PropertyInfoCollection property.

Item

Syntax

`Item`*`(number)`*

Description

The Item method returns a PropertyInfo object that exists at the *number* position in the PropertyInfoCollection.

Parameters

Parameter	*Description*
number	A position in the PropertyInfoCollection collection. The first position is 1.

Returns

A reference to the PropertyInfo object that exists at the *number* position in the PropertyInfoCollection collection. If the specified item is a collection, the returned PropertyInfo object is invalid and is set to NULL.

Example

```
/* Copy all like properties from &Item to &TargetItem */
for &i = 1 to &Item.PropertyInfoCollection.Count
   if &Item.PropertyInfoCollection.Item(&i).IsSimpleApplicationProperty then
     try &TargetItem.SetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name,
        &Item.GetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name))
     catch Exception &xException;

     end-try;
   end-if;
end-for;
```

PropertyInfoCollection Property

The following is the property for the PropertyInfoCollection property.

Count

Description

This property returns the number of PropertyInfo objects in the PropertyInfoCollection .

This property is read-only.

Example

```
/* Copy all like properties from &Item to &TargetItem */
for &i = 1 to &Item.PropertyInfoCollection.Count
   if &Item.PropertyInfoCollection.Item(&i).IsSimpleApplicationProperty then
    try &TargetItem.SetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name,
       &Item.GetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name))
       catch Exception &xException;
     end-try;
   end-if;
end-for;
```

PropertyInfo Class

The PropertyInfo class objects are returned by the following:

- the Item method of a PropertyInfo collection
- the GetPropertyInfoByName of a mobile object

PropertyInfo Class Properties

The following are the properties for the PropertyInfo object.

DefaultValue

Description

This property returns the default value for this mobile object property.

This property is read-only.

Note. Definitional defaults are applied only when an Object is Created, just before the OnInit PeopleCode is run.

Example

```
&pinfo = %ThisMobileObject.GetPropertyInfoByName("QE_MB_SET_DATE");
if &pinfo.HasDefaultValue = True then
   %ThisMobileObject.QE_MB_SET_DATE = &pinfo.DefaultValue;
end-if;
```

HasDefaultValue

Description

This property returns a Boolean value: True if this mobile object property has a default value; False otherwise.

This property is read-only.

Note. Definitional defaults are applied only when an object is created, just before the OnInit PeopleCode is run.

Example

```
&pinfo = %ThisMobileObject.GetPropertyAttrsByName("Text");
if &pinfo.HasDefaultValue = True then;
   %ThisMobileObject.Text = &pinfo.DefaultValue;
end-if;
```

IsCollection

Description

This property returns True if the property that the PropertyInfo object is describing is a Data Collection, False otherwise. If IsCollection is False, the object represents a mobile object user-defined or system property.

This property is read-only.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number field
to determine the item method from the PropertyInfoCollection class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/*Write the results to the results field  */
&qe_mb_obj.QE_MB_RESULTS = "The following is " | &PropertyInfo.IsCollecton | ":⇒
 This Property represents a collection";
```


IsRefToParent

Description

This property is True if this is a reference to this object's parent, False otherwise. A property that is RefToParent is also a System Property.

This property is read-only.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number
field to determine the item method from the PropertyInfoCollection class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/* Write the results to the results field  */
&qe_mb_obj.QE_MB_RESULTS = "The following is " | &PropertyInfo.IsRefToParent | ":⇒
 This is a reference to this Object' Parent. "
```

IsPeerReference

Description

This property is True if this is a peer reference, False otherwise.

This property is read-only.

IsSimpleApplicationProperty

Description

This property is True is this is a simple, non-reference application-defined property; False otherwise. In other words, if True, it is not a system property, collection, nor peer reference; it is a property that was defined on the synchronized Component Interface in Application Designer.

This property is read-only.

Example

```
/* Copy all like properties from &Item to &TargetItem */
for &i = 1 to &Item.PropertyInfoCollection.Count
   if &Item.PropertyInfoCollection.Item(&i).IsSimpleApplicationProperty then
      try &TargetItem.SetPropertyByName(&Item.PropertyInfoCollection.Item⇒
(&i).Name,
        &Item.GetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name))
      catch Exception &xException;

      end-try;
   end-if;
end-for;
```

IsSystemProperty

Description

This property is True if this is a System Property; False otherwise. System properties include the OID and VERSION properties, and references to Parents.

This property is read-only.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number
field to determine the item method from the PropertyInfoCollection class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/*Write the results to the results field  */
&qe_mb_obj.QE_MB_RESULTS = "The following is " | &PropertyInfo.IsSystemProperty |⇒
 ": This is a system property. "
```

Name

Description

The name of the property.

This property is read-only.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number
field to determine the item method from the PropertyInfoCollection class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/*Write the results to the results field  */
&qe_mb_obj.QE_MB_RESULTS = "The name of the accessed Property is: " | &Property⇒
Info.Name;
```

PropertyInfoCollection

Description

This property returns a reference to another PropertyInfoCollection collection if the object executing the property is a collection.

This property is read-only.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number field to determine the item method from the PropertyInfoCollection⇒
 class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/*Write the results to the results field. Since this property returns a CompIntf⇒
PropertyInfoCollect Object, use the Count porperty of the PropertyInfoCollection⇒
 class, to test this.*/
&qe_mb_obj.QE_MB_RESULTS = "The property count of the CompIntfProperyInfo⇒
Collection returned by this property is: " | &PropertyInfo.PropertyInfoCollection⇒
.count;
```

CI Collection Class

A CI Collection is returned by the mobile object Find method. A CI Collection is a collection of empty mobile objects.

See Also

Chapter 26, "Mobile Classes," Find, page 1400

CI Collection Method

The following is the CI Collection method.

Item

Syntax

Item*(index)*

Description

The Item method returns a Mobile Component Interface object. Once you've returned a reference, to instantiate it, use the Get method on the returned object.

Parameters

Parameter	***Description***
index	A number value: the index number for the Component Interface to fetch. Indexes start at 1.

Returns

An empty mobile object that was fetched. You must use the Get method to get the data for that object.

Example

```
/* Error checking left out for clarity */
&MyCI = %Session.GetCompIntfc(CompIntfc.QE_MB_COMPANY_CI);
&MyCI.QE_MB_COMPANY = "Acme Appliances";
&MyCIFindCollection = &MyCI.Find();
&Acme = &MyCIFindCollection.Item(1);
&Acme.Get();
```

CI Collection Properties

The following is the CI Collection property.

Count

Description

The number of Mobile Component Interface objects in the CI Collection collection. This property is read-only.

Example

```
/* Error checking left out for clarity */
&MyCI = %Session.GetCompIntfc(CompIntfc.QE_MB_COMPANY_CI);
&total_in_collection = &MyCI.count;
```

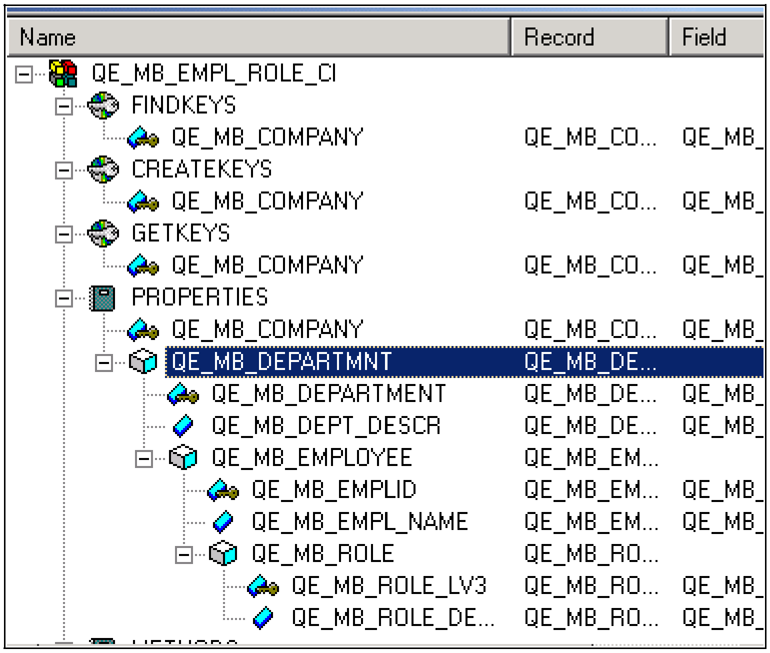
DataCollection Class

A Data Collection object is created when you have a Mobile Page that is based upon a synchronizable Component Interface that contains levels. You do not directly instantiate the DataCollection object.

The synchronizable Component Interface upon which the following Mobile Page is based has the following levels:

- level one: QE_MB_DEPARTMNT
- level two: QE_MB_EMPLOYEE
- level three: QE_MB_ROLE

Use the DataCollection methods and property to access the mobile objects that map to the rows on these levels.



Component Interface Data Collection Levels

Data Collection Methods

Following are the methods for the DataCollection class.

GetListViewAttrs

Syntax

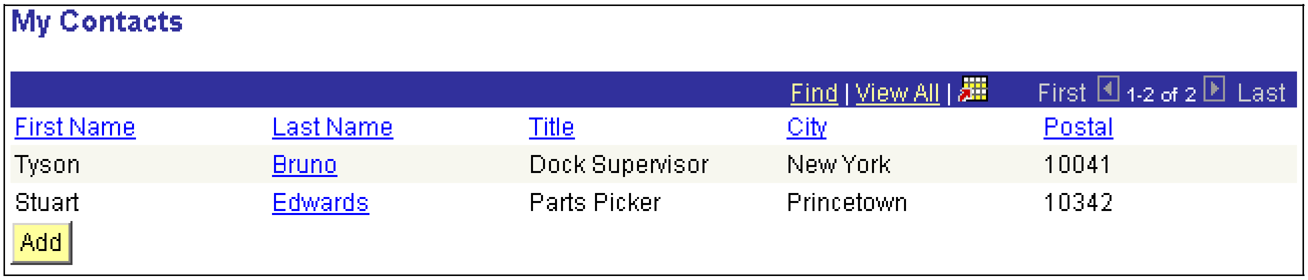
`GetListViewAttrs`*`(propertyname)`*

Description

This method returns the List View attributes for this property. These attributes describe display attributes for a column on a List View.

In the following screen shot, the columns include:

- First Name
- Last Name
- Title
- City
- Postal code



List View columns

Using the GetListViewAttrs method, you can return a reference to one of these columns, then using the ListViewAttrs properties, you can change how the column displays.

Parameters

Parameter	***Description***
propertyname	This is the name of the property for which the List View attributes are retrieved.

Returns

A ListViewAttrs object containing information about the property column.

Example

```
/* Change the Label for the City Property which is a member of the objects
in the QE_DEMO_SITE Collection  to "Town"*/
&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.GetListViewAttrs("City");
&ListAttrs.Label = "Town";
```

Item

Syntax

Item*(index)*

Description

The Index method returns a reference to the mobile object referenced at *index*. This mobile object on the Mobile Page is the equivalent of one row on a level in the synchronizable Component Interface upon which the Mobile Page is based.

Parameters

Parameter	*Description*
index	Specify the index number for the mobile object to return. Indexes start at 1.

Returns

A reference to the mobile object that was returned.

Example

```
Local ApiObject &ob, &Session;
Local ApiObject &PropertyInfo;

&ob = %ThisMobileObject;
/************************************************
* Insert some children                          *
************************************************/
&ChildCount = &ob.MyChild1.Count;

&ob.MyChild1.InsertItem(&ChildCount);
&ChildObject = &ob.MyChild1.Item(&ChildCount + 1);

/************************************************
* Manipulate Collections                        *
************************************************/
&ob.MyChild1.InsertItem(0);
&ChildObject = &ob.MyChild1.Item(1);
&ChildCount = &ob.MyChild1.Count;
If (&ChildCount > 0) Then;
   For &index = 1 To &ChildCount
     &ChildObject = &ob.MyChild1.Item(&index);
     &PropertyInfo = &ChildObject.GetPropertyInfoByName("version");
     &PropertyInfo.Label = "Version: ";
     &ob.MyChild1.Item(&index).version = &index + 100;
     &ob.MyChild1.Item(&index).GetPropertyInfoByName("version").visible = True;
   End-For;
End-If;
```

DeleteItem

Syntax

DeleteItem*(location)*

Description

The DeleteItem method removes the specified item from the collection and deletes the mobile object. This mobile object on the Mobile Page is the equivalent of one row on a level in the synchronizable Component Interface upon which the Mobile Page is based.

Note. Once you delete an item, it is deleted immediately. You no longer have access to it.

Parameters

Parameter	*Description*
location	Specify the location within the collection of the mobile object to delete. This parameter takes a numeric value.

Returns

A Boolean value: True if the mobile object was successfully deleted, False otherwise.

Example

```
Local ApiObject &ob, &Session;
Local ApiObject &PropertyInfo;

&ob = %ThisMobileObject;
/************************************************
* Insert some children                          *
************************************************/
&ChildCount = &ob.MyChild1.Count;

&ob.MyChild1.InsertItem(&ChildCount);
&ChildObject = &ob.MyChild1.Item(&ChildCount + 1);

/************************************************
* Manipulate Collections                        *
************************************************/
&ob.MyChild1.DeleteItem(1);
&ChildObject = &ob.MyChild1.Item(1);
&ChildCount = &ob.MyChild1.Count;
```

InsertItem

Syntax

InsertItem*(location)*

Description

The InsertItem method inserts a new mobile object item at the specified location, plus 1. This mobile object on the Mobile Page is the equivalent of one row on a level in the synchronizable Component Interface upon which the Mobile Page is based.

Parameters

Parameter	***Description***
location	Specify the location of the new item. The mobile object is inserted at one past the number specified with *location.* For example, InsertItem(0) places the item in the first location, which is 1. InsertItem(count) places the new item in the last location: count + 1.

Returns

A Boolean value: True if the mobile object was successfully inserted, False otherwise.

Example

```
Local ApiObject &ob;

&ob = %ThisMobileObject;


If &MobileObjectValueExists = False Then
   &qe_mb_empl.create();
   /*Populate Level 1 data.  */
   &NewCICounter_LV1 = 0;
   &ob_top_parent = &ob.gettopparent();
   While &NewCICounter_LV1 <> &ob_top_parent.QE_MB_DEPARTMNT.count;
      /*Insert Level 1 Row  */
       &qe_mb_empl.QE_MB_DEPARTMNT.InsertItem(&NewCICounter_LV1 + 1);
       &qe_mb_empl.QE_MB_DEPARTMNT.Item(&NewCICounter_LV1 + 1).QE_MB_DEPARTMENT =
       &ob_top_parent.QE_MB_DEPARTMNT.Item(&NewCICounter + 1).QE_MB_DEPARTMENT_LV1;
       &qe_mb_empl.QE_MB_DEPARTMNT.Item(&NewCICounter_LV1 + 1).QE_MB_DEPT_DESCR =
       &ob_top_parent.QE_MB_DEPARTMNT.Item(&NewCICounter_LV1 + 1).QE_MB_DEPT_DESCR;
       &NewCICounter_LV2 = 0;
    End-While;
End-If;
```

Data Collection Properties

Following is the property for DataCollection.

Count

Description

The number of mobile objects in the collection. This is the equivalent of the number of rows on a level in the synchronizable Component Interface upon which the Mobile Page is based.

This property is read-only.

Example

```
Local ApiObject &ob, &Session;
Local ApiObject &PropertyInfo;

&ob = %ThisMobileObject;
/************************************************
* Insert some children                          *
************************************************/
&ChildCount = &ob.MyChild1.Count;

&ob.MyChild1.InsertItem(&ChildCount + 1);
&ChildObject = &ob.MyChild1.Item(&ChildCount + 1);

&ob.MyChild1.InsertItem(1);
&ChildObject = &ob.MyChild1.Item(1);
&ChildCount = &ob.MyChild1.Count;
```

ListViewAttrs Class

The ListViewAttrs class objects are returned from the GetListViewAttrs data collection method.

The ListViewAttrs class has methods and properties that provide information about the properties for the synchronizable Component Interface upon which the Mobile Page is based.

ListViewAttrs Method

Following is the ListViewAttrs method.

IsSet

Syntax

IsSet*(attribute)*

Description

The IsSet method indicates if the given attribute had been explicitly set from PeopleCode, or if it is in an un-initialized state.

Parameters

Parameter	*Description*
attribute	Specify one of the ListViewAttrs properties, which contains display information about a mobile property column in a List View.

Returns

This method returns a Boolean value: True if this attribute has been set, False otherwise.

Example

```
/* Write the results to the results string  */

&results_string = "The following is " |
&ThisMobileObject.QE_DEMO_SITE("City").IsSet("Visible") |
": City is visible. "
```

ListViewAttrs Properties

Following are the ListViewAttrs properties.

DetailViewLabel

Description

Use this property to change the label shown at the top of a Mobile Page Detail View. DetailViewLabel is only available to a ListViewAttrs object that has been fetched with an empty string.

This property is read-write.

Example

Note that in this example, GetListViewAttrs uses an empty string as input. This is how to call GetListViewAttrs for use with DetailViewLabel.

```
/* Pass empty string */
&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.Item(1).QE_DEMO_SITE_H.⇒
GetListViewAttrs("");
&ListAttrs.DetailViewLabel = "My Site Details";
```

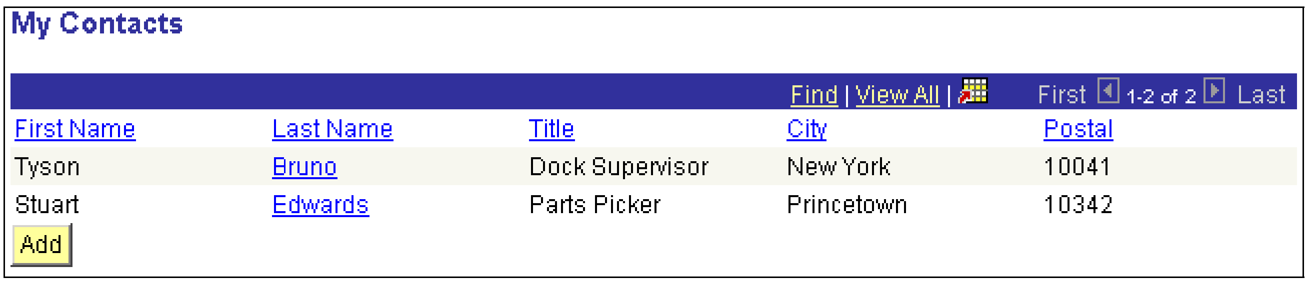
Label

Description

Use this property to change the label shown at the top of a column in a List View; it is the label for a property for mobile objects in a Data Collection.

This property is read-write.

In the following example, there are five columns, hence five labels: First Name, Last Name, Title, City and Postal.



List View columns

Example

```
/* Change the Label for the City Property which is a member of the objects
in the QE_DEMO_SITE Collection to "Town"*/
&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.GetListViewAttrs("City");
&ListAttrs.Label = "Town";
```

ListViewLabel

Description

Use this property to change the tab label. The tab label is indirectly exposed through PeopleCode; it overwrites the List View Label, which is the default if no tab label is defined. However, if the ListViewLabel property is set for a property that has a tab label, the tab label is overridden with the PeopleCode ListViewLabel value.

ListViewLabel is only available to a ListViewAttrs object that has been fetched with an empty string.

This property is read-write.

Example

Note that in this example, GetListViewAttrs uses an empty string as input. This is how to call GetListViewAttrs for use with ListViewLabel.

```
/* Pass empty string */
&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.Item(1).QE_DEMO_SITE_H.⇒
GetListViewAttrs("");
&ListAttrs.ListViewLabel = "My Sites";
```

Visible

Description

This property is True if this column for a property in a List View is visible. Setting this property to False hides the column.

Note. Even if a property has been marked as Invisible on the mobile page definition in Application Designer, you can still override this property using PeopleCode.

This property is read-write.

Example

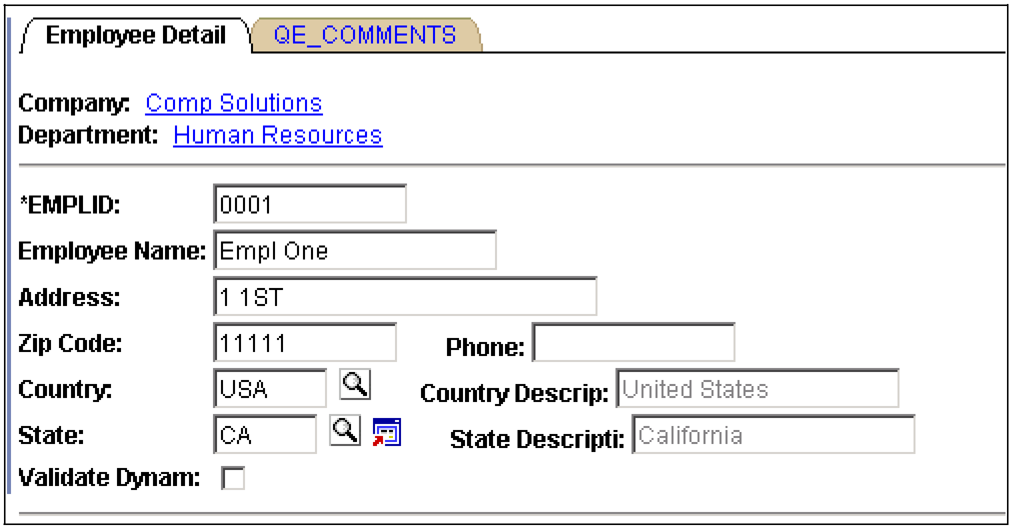
```
/* Make invisible the City property column which is a member of
the objects in the QE_DEMO_SITE Collection */

&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.GetListViewAttrs("City");
&ListAttrs.Visible = False;
```

PropertyAttrs Class

The PropertyAttrs class objects are returned from the GetPropertyAttrsByName method. This class has methods and properties that provide information about the properties for the synchronizable Component Interface upon which the Mobile Page is based.

For example, to hide the QE_MB_ADDRESS property on the synchronizable Component Interface upon which the current Mobile Page and its mobile objects are based, you can use the following PeopleCode on an OnChange event.

```
%ThisMobileObject.GetPropertyAttrsByName("QE_MB_ADDRESS").Visible = False;
```

Mobile Page Detail View

PropertyAttrs Method

Following is the PropertyAttrs method.

IsSet

Syntax

`IsSet``(`*`attribute`*`)`

Description

The IsSet method indicates if the given attribute had been explicitly set, or if it is in an un-initialized state.

Parameters

Parameter	***Description***
attribute	Specify one of the PropertyAttrs properties, which contains display information about a mobile property.

Returns

This method returns a Boolean: True if this attribute has been set, False otherwise.

Example

```
/*Write the results to the results field  */

&qe_mb_obj.QE_MB_RESULTS = "The following is " |
&qe_mb_obj_setup.GetPropertyAttrsByName(&CharacterPropertyName).IsSet("Visible") |
": " | GetPropertyAttrsByName(&CharacterPropertyName) | "is visible. ";
```

PropertyAttrs Properties

Following are the PropertyAttrs properties.

DisplayOnly

Description

This property is True if the mobile property is display only; False otherwise. This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

Note. If a property has been marked as Display Only on the mobile page definition in Application Designer, you can override this property using PeopleCode. If a property has been marked as Display Only on the Component Interface, you can *not* override this property using PeopleCode.

If you use this property to make a collection display only, all the fields will be display only, except for those that are displayed as pushbuttons. To make a pushbutton display only, you must set the pushbutton property specifically to be display only, either in Application Designer or with PeopleCode. You must set the pushbutton property for every row to be display only.

This property is read-write.

Example

```
Declare Function Hide_or_Unhide_WorkingSetProps_SetValues PeopleCode QE_MB_PC_⇒
DER.QE_MB_GET FieldFormula;

/*If Get() checkbox is selected, unhide Set Value Properties, and grey
Create Check Box. Also maske sure Get() check box is ungrayed. */
If %ThisMobileObject.QE_MB_GET = True Then
   /*Make sure Create() Checkbox now has value "N"  */
   %ThisMobileObject.QE_MB_CREATE = False;
   Hide_or_Unhide_WorkingSetProps_SetValues( True);
   %ThisMobileObject.GetPropertyAttrsByName("QE_MB_CREATE").DisplayOnly = True;

Else
   /* If Get() checkbox is unselected, hide Set Value Properties, and ungrey
   Create Check Box. */
   If %ThisMobileObject.QE_MB_GET = False Then
      Hide_or_Unhide_WorkingSetProps_SetValues( False);
      %ThisMobileObject.GetPropertyAttrsByName("QE_MB_CREATE").DisplayOnly = False;
   End-If;
End-If;
```

InError

Description

Set this property within an OnSaveEdit event to cause any field on a Detail View that contains incorrect information to be displayed as red. This alerts the user to correct the data in that field.

This field is automatically reset. The developer does not have to reset this field.

This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

This property is read-write.

Example

```
%ThisMobileObject.GetPropertyAttrsByName("QE_MB_ADDRESS").InError = True;
```

Label

Description

Use this property to manipulate the label for this mobile property. This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

This property is read-write.

Example

```
&ob = %ThisMobileObject;
&PropertyInfo = &ob.GetPropertyAttrsByName("text");
&PropertyInfo.Label = "This is text";
```

ShowRequiredCue

Description

An asterisk (*) is displayed beside fields that are defined as Required in Application Designer. You can use this property to specify whether this asterisk, also called the required field cue, is displayed for a particular field.

This property is True if the mobile property displays the RequiredCue; False otherwise. This only affects the Mobile Page; this is not saved to the mobile database and is not synchronized.

Use this to make a field be required that was not specified as being required when the synchronized Component Interface was created in the Application Designer. (You cannot make a required field be not required.)

For example, many fields are made required through PeopleCode. This means they aren't defined as Required in Application Designer, and the end user may be confused. For these fields, you can use this property.

Note. This property affects only fields where a required field cue is otherwise permissible. That is, regardless of the setting of the property, no cue is ever shown on a push button, a display-only field, and so on.

This property is read-write.

Example

```
&ob = %ThisMobileObject;
&PropertyInfo = &ob.GetPropertyAttrsByName("text");
&PropertyInfo.ShowRequiredCue = True;
```

Visible

Description

This property is True if this field is visible in the Mobile Page Detail View displaying it. Setting this property to False hides the field.

This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

Note. Even if a property has been marked as Invisible on the mobile page definition in Application Designer, you can still override this property using PeopleCode.

If you set this value to False for a collection property, the collection is no longer visible.

Note. You can only make collections invisible using PeopleCode. You cannot set a collection to be invisible in Application Designer.

This property is read-write.

Example

```
If %ThisMobileObject.QE_MB_PROP_METH = "Visible" Or
   %ThisMobileObject.QE_MB_PROP_METH = "DisplayOnly" Or
   %ThisMobileObject.QE_MB_PROP_METH = "ShowRequiredCue" Or
   %ThisMobileObject.QE_MB_PROP_METH = "SetModified" Then
   %ThisMobileObject.getpropertyattrsbyname("QE_MB_TRUE_FALSE").Visible = True;
End-If;

If %ThisMobileObject.QE_MB_PROP_METH = "Item" Or
   %ThisMobileObject.QE_MB_PROP_METH = "DeleteItem" Or
   %ThisMobileObject.QE_MB_PROP_METH = "InsertItem" Then
   %ThisMobileObject.getpropertyattrsbyname("QE_MB_PC_ITEM").Visible = True;
End-If;
```

PeerDefaultAttributes Class

PeerDefaultAttributes are returned by the GetPeerDefaultAttributesByName method of a mobile object.

When a peer reference does not have a value associated with it, the display attributes can only be accessed using the PeerDefaultAttributes class. After a peer reference has values, it can access using the ListViewAttrs class.

PeerDefaultAttributes Class Method

The following is the PeerDefaultAttributes class method.

IsSet

Syntax

IsSet*(Attribute)*

Description

The IsSet method indicates if the given attribute had been explicitly set, or if it is in an un-initialized state.

Parameters

Parameter	Description
Attribute	Specify one of the PeerDefaultAttributes properties.

Returns

A Boolean value: True if the attribute has been explicitly set, False otherwise.

Example

```
&PeerAtt = %ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB",⇒
 "ADDRESS");

If &PeerAtt.IsSet(Visible) Then

/* some code */

End-if;
```

PeerDefaultAttributes Properties

This section discusses the PeerDefaultAttributes properties in alphabetical order.

DisplayOnly

Description

This property is True if the peer reference is display only; False otherwise. This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

Note. If a property has been marked as Display Only on the mobile page definition in Application Designer, you can override this property using PeopleCode. If a property has been marked as Display Only on the Component Interface, you can *not* override this property using PeopleCode.

This property is read-write.

Example

```
&IsVisible =
%ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB", "ADDRESS").⇒
DisplayOnly;

If &DisplayOnly Then

/* some code */

Else

/* some other code */

End-if;
```

Label

Description

Use this property to manipulate the label for this peer reference. This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

This property is read-write.

Example

```
%ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB", "ADDRESS").⇒
Label = &Label;
```

Visible

Description

This property is True if this peer reference is visible in the Mobile Page Detail View displaying it. Setting this property to False hides the peer reference.

This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

This property is read-write.

Example

```
%ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB", "ADDRESS").⇒
Visible = True;
```

Classes and Functions and System Variables Used in Mobile

Mobile uses a subset of the classes, functions, and system variables used in PIA. Here is a list of those used in mobile.

PeopleCode Classes Used in Mobile

The following is a list of the PeopleCode classes used in mobile.

- Application Package
- Array
- Component Interface
- Exception
- Request (part of iScript)

Considerations Using the Request Class

Only the following request class method (%Request) is supported for mobile:

GetParameter

Only the following request class properties (%Request) are supported for mobile:

- BrowserType
- BrowserVersion
- HTTPMethod
- Protocol

Note. Other request class methods and properties may not return an error when used in mobile, but do return a blank value, an empty array, False, and so on, according to the datatype of the value returned.

See Chapter 21, "Internet Script Classes (iScript)," Request Class Methods, page 1062.

System Variables Used in Mobile

The following is a list of the system variables used in mobile, by category.

- APIs: %Session, %Super, %This, %CompIntfcName
- Data and Time: %Date, %DateTime, %Request, %Time
- Mobile specific: %DeviceType, %MobilePage, %SyncServer, %ThisMobileObject

Considerations Using Default Values

For a record field, you can specify constants for a default, such as %Date. Mobile only supports the following for record field defaults. In addition, these are only supported for date, datetime, and time type fields (as appropriate):

- %ClientDate
- %Date
- %DateTime
- %Time

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "System Variables"

Built-In Functions Used in Mobile

The following is a list of the built-in functions used in mobile, by category.

- Arrays: CreateArray, CreateArrayAny, CreateArrayRept, Split
- Conversion: Char, Code, String, Value
- Date and Time: AddToDate, AddToDateTime, AddToTome, Date, Date3, DatePart, DateTime6, DateTimeValue, DateValue, Day, Days, Days360, Days365, Hour, Minute, Month, Second, Time, Time3, TimePart, TimeValue, Weekday, Year
- Internet: GetHTMLText
- Language Constructs: Break, Declare Function, Evaluate, Exit, For, Function, If, Local, Repeat, Return, While
- Math: Abs, Acos, Asin, Atan, Cos, Cot, Degrees, Exp, Fact, Int, Integer, Ln, Log10, Nod, Product, Radians, Rand, Round, Sign, Sin, Sqrt, Tan, Truncate
- Message Catalog and Message Display: MessageBox, MsgGet, MsgGetExplainText, MsgGetText
- Mobile: GenerateMobileTree, IsDisconnectedClient, MSFGetNextNumber, TransferMobilePage
- OLE: CreateObject, ObjectDoMethod, ObjectGetProperty, ObjectSetProperty
- Saving: DoSave

  Note. DoSave can only be used in the OnChange and OnChangeEdit events.

- String: Clean, Code, Exact, Find, Left, Len, Lower, LTrim, Replace, Rept, Right, Rtrim, String, Substitute, Substring, Upper
- Validation: Error
- Workflow: Warning

Considerations Using DoSave

The DoSave built-in function is supported in the following Mobile events:

- OnChange
- OnChangeEdit

Save processing (including OnSaveEdit, OnSavePreChange, and OnSavePostChange) is executed exactly as if the end user clicked the save button. OnObjectChangeEdit and OnObjectChange are executed prior to initiating the save processing.

If an error occurs during save processing, it is handled the same as if the user clicked the save button. For example, if save processing encounters an OnSaveEdit error, it displays the appropriate message box. If the error is on the current detail view, the detail view is rendered with the error highlighted. If the error is not on the current detail view, the user is asked if they would like to transfer to the object with the error. The user can correct the error and continue working with the component. If the PeopleCode calls DoSave followed by TransferMobilePage and an error occurs during save processing, the TransferMobilePage is not executed.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions"

Chapter 27

Notification Classes

This chapter provides an overview of the Notify classes and discusses the following topics:

- Scope of the notification classes.
- Data types of the notification classes.
- How to import the notification classes.
- How to create a notification object.
- Notification class constructors.
- Notification class reference.

Understanding Notification Classes

The Notification classes are used for ad-hoc notification. These classes enable you to create and send a notification to someone. The notifications can either be a Peoplesoft Worklist item or an SMTP email message. The Notification class also stores notifications in a table.

You can also use web services to create Worklist entries, then send notifications of completion using application messages.

The Notification classes can be called from a component, an internet script, or an Application Engine program. The WSWorklistEntry class can be used incase you are developing your own PT_WORKLIST notification handler. .

Most of the classes, as well as most of the properties and methods that make up the Notification classes have a GUI representation in Peoplesoft Workflow. This document assumes that the reader is familiar with PeopleSoft Workflow.

See Also

Enterprise PeopleTools 8.51 PeopleBook: Workflow Technology, "Understanding PeopleSoft Workflow"

Scope of the Notification Classes

The Notification classes can be called from a component, an internet script, Integration Broker, or an Application Engine program.

Notification classes can be of Local, Global, or Component scope.

Data Types of the Notification Classes

Every Notification class is its own data type, that is, Notifications are declared as data type Notification, Notification addresses are declared as type NotificationAddress, and so on.

The following are the data types of the Notification classes:

- Notification
- NotificationAddress
- NotificationTemplate
- WorklistEntry
- Worklist
- WSWorkListEntry

How to Import the Notification Classes

The Notification classes are *not* built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing the Notification classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_WF_NOTIFICATION contains the following classes:

- Notification
- NotificationAddress
- NotificationTemplate

The application package PT_WF_WORKLIST contains the following classes:

- Worklist
- WorklistEntry
- WSWorkListEntry

The import statements you should use are as follows:

```
import PT_WF_NOTIFICATION:*;
import PT_WF_WORKLIST:*;
```


Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available.

See Also

Chapter 6, "Application Classes," page 189

How to Create a Notification Object

After you've imported the Notification classes, you instantiate an object of one of those classes using the constructor for the class and the Create function.

The following example creates a new instance of the WorklistEntry class, as the variable &MyWE, with local scope:

```
Local WorklistEntry &MyWE = Create WorklistEntry();
```

See Also

Chapter 27, "Notification Classes," Notification Classes Constructors, page 1447

Notification Classes Constructors

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the Notification classes.

Notification

Syntax

Notification*(NotifyFrom, dttmCreated, language_cd)*

Description

Use Notification to instantiate a notification object.

Parameters

Parameter	***Description***
NotifyFrom	Specify the entity the notification is being sent from, as a string. Avoid overusing generic entities with this parameter, as this value, along with the value for *dttmCreated* are the key to understanding where notifications originated from.
dttmCreated	Specify the date and time the notification is being sent, as a DateTime value. **Important!** To ensure a unique timestamp for each notification that is sent, you must pass a DateTime value that includes milliseconds. Because the %Datetime system variable always returns 0 for the milliseconds value, use the combination of %Date+%PerfTime instead.
language_cd	Specify the 3-character language code of the current user, as a string.

Returns

A Notification object.

Example

To ensure a unique timestamp for each notification that is sent, you must pass a DateTime value that includes milliseconds to the *dttmCreated* parameter. Because the %Datetime system variable always returns 0 for the milliseconds value, use the combination of %Date+%PerfTime instead.

```
import PT_WF_NOTIFICATION:*;
import PT_WF_WORKLIST:*;

Local PT_WF_NOTIFICATION:Notification &MyNotify;

&MyNotify = create PT_WF_NOTIFICATION:Notification("Workflow_Admin", ⇒
%Date + %PerfTime, %Language);
```

See Also

Chapter 27, "Notification Classes," Notification Class, page 1453

NotificationAddress

Syntax

`NotificationAddress`*`(Oprid,Description,Language,EmailId,Channel)`*

Description

Use NotificationAddress to instantiate a notification address.

This address is used to build a notification using the Notification class. This class is required if you want to use the Notification data type.

Parameters

Parameter	***Description***
Oprid	Specify the user ID of the person receiving the notification, as a string.
Description	Specify the description of the entity receiving the notification (distribution group names, people's names, and so on) as a string.
Language	Specify the language code of the entity to be notified as a string.
EmailId	Specify the email address as a string.
Channel	Specify the 'channel' used for the notification, as a string. The values are:

Value	***Description***
Email	Notification is sent as email
Worklist	Notification is sent as a worklist

Returns

None.

Example

```
import PT_WF_NOTIFICATION:*;

import PT_WF_WORKLIST:*;

Local NotificationAddress &MyNotify_Addy;

&MyNotify_Addy = Create NotificationAddress(%UserID, &Description, %Language,⇒
 &EmailID, &Channel);
```

See Also

Chapter 27, "Notification Classes," NotificationAddress Class, page 1459

NotificationTemplate

Syntax

`NotificationTemplate``(ComponentId, Market, TemplateId, TemplateType)`

Description

Use NotificationTemplate to instantiate a notification template.

If you're specifying a generic template, the first two parameters do not have to have values, that is, you must specify a Null string ("") instead.

Parameters

Parameter	***Description***
ComponentId	Specify the name of the component the notification is coming from as a string. To specify a generic template, you must specify a Null string ("") for this value instead.
Market	Specify the name of the market of the component the notification is coming from as a string. To specify a generic template, you must specify a Null string ("") for this value instead.
TemplateId	Specify the template ID as a string.
TemplateType	Specify the type of template as a string. Values are: • G: generic template • C: component template

Returns

An instance of the NotificationTemplate class.

Example

```
import PT_WF_NOTIFICATION:*;

import PT_WF_WORKLIST:*;

Local NotificationTemplate &Notify_Temp;

/* create a generic template */

&Notify_Temp = Create NotificationTemplate("", "", "MyTemplate", "G");
```

See Also

Chapter 27, "Notification Classes," NotificationTemplate Class, page 1460

Worklist

Syntax

Worklist()

Description

Use the Worklist method to instantiate a Worklist object.

Parameters

None.

Returns

An instance of the Worklist class.

Example

The following example code creates a worklist.

```
import PT_WF_WORKLIST:*;

Local Worklist &theWorklist = Create Worklist();
```

See Also

Chapter 27, "Notification Classes," Worklist Class, page 1466

WorklistEntry

Syntax

WorklistEntry()

Description

Use WorklistEntry to instantiate a WorklistEntry object.

Parameters

None.

Returns

An instance of the WorklistEntry class.

Example

```
import PT_WF_NOTIFICATION:*;

import PT_WF_WORKLIST:*;

Local WorklistEntry &MyWorklistEntry = Create WorklistEntry();
```

See Also

Chapter 27, "Notification Classes," WorklistEntry Class, page 1467

WSWorklistEntry

Syntax

WSWorklistEntry()

Description

Use the WSWorklistEntry constructor to instantiate a WSWorklistEntry object.

The WSWorklistEntry class is primarily used with web services.

Parameters

None.

Returns

A reference to a WSWorklistEntry object.

See Also

Chapter 27, "Notification Classes," WSWorklistEntry Class, page 1482

Notification Class

This class is used to send a notification to people or people entities, as channel independent as possible. This enables functionality such as letting a user determine how to receive workflow notifications as a preference. It does not necessarily limit a person or entity to one channel.

See Chapter 27, "Notification Classes," Notification Classes Examples, page 1484.

See *Enterprise PeopleTools 8.51 PeopleBook: Workflow Technology*, "Using Notification Templates."

Notification Class Import Statements

The Notification class imports the following classes:

- NotificationAddress
- WorklistEntry

Here are the import statements:

```
import PT_WF_NOTIFICATION:NotificationAddress;

import PT_WF_WORKLIST:WorklistEntry;
```

Notification Class Method

In this section, we discuss the Notification class method Send.

Send

Syntax

Send()

Description

Use the Send method to send a notification to the entities defined in the NotifyTo, NotifyCC, and NotifyBCC, using the Subject and Message properties.

Parameters

None.

Returns

None.

See Also

Enterprise PeopleTools 8.51 PeopleBook: Workflow Technology, "Understanding PeopleSoft Workflow"

Notification Class Properties

In this section, we discuss the Notification class properties. The properties are discussed in alphabetical order.

ContentType

Description

This property returns the content type of the value specified by the Message property, as a string. This property is used only with email Notifications, that is, with SMTP email.

This property is read-write.

See Also

Chapter 27, "Notification Classes," Message, page 1456

dttmCreated

Description

This property returns the Datetime the notification created as a datetime value. Together with NotifyFrom this makes a notification unique.

See Also

Chapter 27, "Notification Classes," NotifyFrom, page 1457

EmailReplyTo

Description

If an email is sent as a result of the Send method, this property returns the email reply that should be used, as a string. This property can have a different value from NotifyFrom. This property is valid only with notifications that have "email" specified as the channel.

This property is read-write.

See Also

Chapter 27, "Notification Classes," Send, page 1453 and Chapter 27, "Notification Classes," NotifyFrom, page 1457

FileNames

Description

This property returns an array of string, populated with the names of any file attachments sent with the notification. This property is valid only with notifications that have "email" specified as the channel.

This property is read-write.

FileTitles

Description

This property returns an array of string, populated with descriptions of any file attachments. The array order *must* be respective to the array order for FileNames, that is, file description matches file name one by using the same array index value. This property is valid only with notifications that have "email" specified as the channel.

This property is read-write.

See Also

Chapter 27, "Notification Classes," FileNames, page 1455

language_cd

Description

This property returns the 3-character language code used for the notification, as a string. This is used for tracking purposes.

This property is read-write.

Message

Description

This property returns the text of the message sent with the notification as a string.

This property is read-write.

NotifyBCC

Description

This property returns an array of NotificationAddress objects, populated with the email addresses or worklist users of the people the notification is sent to. This list is used for notification information only, that is, the people on this list are generally not responsible for taking action specified by the notification. In addition, when you use the Notification Send method, other recipients cannot see the email addresses and or users that make up this array.

If you want to send a regular CC list, that is, one where the members of the list can see each other, use the NotifyCC property instead.

See Also

Chapter 27, "Notification Classes," NotifyCC, page 1456

NotifyCC

Description

This property returns an array of NotificationAddress objects, populated with the email addresses of the people the notification is sent to. This list is used for notification information only, that is, the people on this list are generally not responsible for taking action specified by the notification. When using the Notification Send method, the email address or worklist users of this array are exposed to the recipients of the notification.

If you want to send a blind-CC list, that is, where the members of the list cannot see each other, use the NotifyBCC property instead.

See Also

Chapter 27, "Notification Classes," NotifyBCC, page 1456

NotifyFrom

Description

This property returns the ID of the entity sending the notification as a string. For example, ad-hoc workflow uses the current user ID.

This property is read-write.

NotifyGuid

Description

This property returns the GUID of the notification. This is a unique identifier used for tracking purposes only.

This property is read-write.

NotifyTo

Description

This property returns an array of NotificationAddress objects, populated with the email addresses of the people the notification is sent to, and who are responsible for taking action if necessary.

To send the list to additional people, use the NotifyCC or NotifyBCC properties.

See Also

Chapter 27, "Notification Classes," NotifyCC, page 1456 and Chapter 27, "Notification Classes," NotifyBCC, page 1456

Rte

Description

Use this property to set or return a Boolean value indicating whether to use Rich Text formatting in the notification.

This property is read-write.

SourceComponent

Description

This property returns the name of the source component, that is, that initiated the notification process. This property is used for transaction tracking purposes only. This property is not required.

This property is read-write.

SourceMarket

Description

This property returns the name of the market of the source component, that is, that initiated the notification process. This property is used for transaction tracking purposes only. This property is not required.

This property is read-write.

SourceMenu

Description

This property returns the name of the menu of the source component, that is, that initiated the notification process. This property is used for transaction tracking purposes only. This property is not required.

This property is read-write.

Subject

Description

This property returns the subject used in the email notification as a string.

This property is read-write.

Template

Description

If the notification message originated from a template, this property returns the name of that template, as a string. This property is used for transaction tracking purposes only. This property is not required.

This property is read-write.

NotificationAddress Class

The main purpose of the NotificationAddress is to define an entity to a 'channel', it is concerned at a high level only that a notification was sent.

See Chapter 27, "Notification Classes," Notification Classes Examples, page 1484.

See *Enterprise PeopleTools 8.51 PeopleBook: Workflow Technology*, "Using Notification Templates."

NotificationAddress Class Properties

In this section, we discuss the NotificationAddress class properties. The properties are discussed in alphabetical order.

Channel

Description

This property returns the 'channel' used for the notification, as a string. The values are:

Value	***Description***
Email	Notification is sent as email.
Worklist	Notification is sent as a worklist.

This property is read-write.

Description

Description

This property returns the description of the entity as a string. Entity descriptions include distribution group names or people's names.

This property is read-write.

EmailId

Description

This property returns the Email Address of the entity to be notified.

This property is read-write.

Language

Description

This property returns the language code of the entity to be notified.

This property is read-write.

Oprid

Description

This property returns the user ID of the person receiving the notification as a string.

This property is read-write.

NotificationTemplate Class

This class is useful for populating the message property of the Notification class.

See *Enterprise PeopleTools 8.51 PeopleBook: Workflow Technology*, "Using Notification Templates."

NotificationTemplate Class Methods

In this section, we discuss the NotificationTemplate class methods. The methods are discussed in alphabetical order.

GetAndExpandTemplate

Syntax

`GetAndExpandTemplate(Language, Vars)`

Description

Use the GetAndExpandTemplate method to expand and map the variables into the Text property of the template.

If this is a component template, you must use the SetupCompVarsAndRcpts method first, before you use this method.

If this is a generic template, you must use the SetupGenericVars method first, before you use this method.

The value returned by the Setup methods must be used as the second parameter for this method.

Parameters

Parameter	***Description***
Language	Specify the 3-character language code used with the notification.
Vars	Specify the variable representing the XML string returned by either the SetupCompVarsAndRcpts method or the SetupGenericVars method.

Returns

Returns a Boolean value: True if the template expanded properly, False otherwise.

Example

```
import PT_WF_NOTIFICATION:NotificationTemplate;

Local NotificationTemplate & mynotifytemplate;

&mynotifytemplate = create NotificationTemplate(%Component, %Market, &template⇒
Name, "C");

&xmlVars = &mynotifytemplate.SetupCompVarsAndRcpts(GetLevel0());

&mynotifytemplate.GetAndExpandTemplate(%Language, &xmlVars);
```

See Also

Chapter 27, "Notification Classes," SetupCompVarsAndRcpts, page 1462 and Chapter 27, "Notification Classes," SetupGenericVars, page 1463

SetupCompVarsAndRcpts

Syntax

SetupCompVarsAndRcpts*(&Rowset_Context)*

Description

Use the SetupCompVarsAndRcpts method to set up variables for the component from the provided rowset. This method also sets up any additional recipients defined in the component.

The rowset corresponds to the component that the notification template was instantiated with. For example, if you created a notification template for JOB, the rowset used with this method should be based on the JOB component.

If you want to expand a component template, you must use this method first, before expanding the template.

Parameters

Parameter	*Description*
&Rowset_Context	Specify an already instantiated and populated rowset containing the necessary variables.

Returns

A string containing XML that is used by the GetAndExpandTemplate method.

Example

```
import PT_WF_NOTIFICATION:NotificationTemplate;

Local NotificationTemplate & mynotifytemplate;

&mynotifytemplate = create NotificationTemplate(%Component, %Market, &template⇒
Name, "C");

&xmlVars = &mynotifytemplate.SetupCompVarsAndRcpts(GetLevel0());

&mynotifytemplate.GetAndExpandTemplate(%Language, &xmlVars);
```

See Also

Chapter 27, "Notification Classes," GetAndExpandTemplate, page 1461

SetupGenericVars

Syntax

`SetupGenericVars``(`*`&AryValues`*`)`

Description

Use the SetupGenericVars method to pass values to a generic template.

To expand a generic template, you must use this method first, before expanding the template.

Parameters

Parameter	*Description*
&AryValues	Specify an array of string containing values used with the expanded template.

Returns

A string containing XML that is used by the GetAndExpandTemplate method.

Example

```
import PT_WF_NOTIFICATION:NotificationTemplate;

. . .

&mynotifytemplate = create NotificationTemplate("","", "MYTEMPLATE", "G");

/* Populate an array to contain the values needed by */
/* the template */
&aryValues = CreateArrayRept("", 0);
&aryValues.Push("FIRST VALUE");
&aryValues.Push("SECOND VALUE");
&xmlVars = &mynotifytemplate.SetupGenericVars(&aryValues);
&mynotifytemplate.GetAndExpandTemplate(%Language, &xmlVars);
```

See Also

Chapter 27, "Notification Classes," GetAndExpandTemplate, page 1461

NotificationTemplate Class Properties

In this section, we discuss the NotificationTemplate class properties. The properties are discussed in alphabetical order.

ComponentId

Description

This property returns a component for the component type notification template as a string. This is not required for generic templates.

This property is read-write.

Instruction

Description

This property returns the notification instructions as a string.

This property is read-write.

Language

Description

This property returns the language used to retrieve template descriptions, as a string.

This property is read-write.

Market

Description

This property returns the market for component type notification templates as a string. This is not required for generic templates.

This property is read-write.

Priority

Description

This property returns the priority of the notification template as a string.

This property is read-write.

Responses

Description

This property is populated only if there are any RIMM responses.

This property is read-write.

Subject

Description

This property returns the subject of the notification template as a string.

This property is read-write.

TemplateId

Description

This property returns the template ID as a string. If you are creating a notification template, this property is always required.

This property is read-write.

TemplateType

Description

This property returns the template type as a string. Values are:

Value	***Description***
C	Component
G	Generic

If you are creating a notification template, this property is always required.

This property is read-write.

Text

Description

This property returns the Template Text as a string.

This property is read-write.

Worklist Class

The Worklist class is used to make mass modifications to worklist instances. Modifications to one worklist instance should be performed through the WorklistEntry class.

See Chapter 27, "Notification Classes," Worklist, page 1451.

Worklist Class Method

In this section, we discuss the Worklist class method Reassign.

Reassign

Syntax

`Reassign``(`*`FromOperid`*`,`*`ToOperid`*`)`

Description

Use the Reassign method to reassign all worklist items with a state less than "Worked" from one user to another.

If you only want to reassign a single worklist entry, use the Reassign WorklistEntry class method.

Parameters

Parameter	***Description***
FromOperid	Specify the user ID of the user from which you want to assign a worklist, as a string.
ToOperid	Specify the user ID of the user to which you want to assign a worklist, as a string.

Returns

A Boolean value: true if worklist was assigned successfully, false otherwise.

See Also

Chapter 27, "Notification Classes," Reassign, page 1470

WorklistEntry Class

This class is used to create and or update a worklist entry.

See Chapter 27, "Notification Classes," Creating a WorklistEntry, page 1484.

See Chapter 27, "Notification Classes," Updating a WorklistEntry, page 1486.

See *Enterprise PeopleTools 8.51 PeopleBook: Workflow Technology*, "Understanding PeopleSoft Workflow."

WorklistEntry Class Methods

In this section, we discuss the WorklistEntry class methods. The methods are discussed in alphabetical order.

Create

Syntax

`Create``()`

Description

Use the Create method to create a new WorklistEntry object.

This method is *not* the same as the Create *constructor.* The Create constructor only creates an instance of the WorklistEntry class. To add the entry to the database, you must use this method.

The following properties must be set before calling this method:

- busactivity
- buseventname
- busprocname
- worklist

If this method completes successfully, the following properties are populated:

- instanceid
- transactionid

Parameters

None.

Returns

This method returns a numeric value: 0 if there is an error, nonzero if entry created successfully.

Example

```
import PT_WF_WORKLIST:*;

Local WorklistEntry &worklist;
Local Number &Rslt;

&worklist = create WorklistEntry();

&worklist.busprocname  = "Administer Workflow";
&worklist.busactivity  = "Send Note";
&worklist.buseventname = "Worklist Note";
&worklist.worklistname = "Worklist Note";

&Rslt = &worklist.Create();
```

See Also

Chapter 27, "Notification Classes," busactivity, page 1476; Chapter 27, "Notification Classes," buseventname, page 1476; Chapter 27, "Notification Classes," busprocname, page 1476; Chapter 27, "Notification Classes," instanceid, page 1477 and Chapter 27, "Notification Classes," transactionid, page 1480

GetResponseStatus

Syntax

GetResponseStatus()

Description

Use the GetResponseStatus method to return the response status of the callback method. This is useful only for entries that are created by web services.

The following properties must be set before calling this method:

- busactivity
- buseventname
- busprocname
- requestmessageid
- worklistname

Parameters

None.

Returns

A number. Valid values are:

Value	*Description*
0	CreateWorklistEntry message not received yet or this entry was not created by any web service.
1	Message received but not processed yet.
2	Message processed and sent to message queue.
3	Error while sending message to the caller of create worklist entry.
4	Message was sent to the originating web server and was received successfully.

Reassign

Syntax

`Reassign``(`*`Operid`*`)`

Description

The Reassign method assigns the worklist in the data buffers to the user specified by *Operid.* This method commits the version of the worklist entry in the data buffer to the database using the Save method.

If you want to reassign an entire worklist, use the Reassign Worklist class method.

Note. You must use the SelectByKey method to populate data buffers with the worklist entry.

Parameters

Parameter	*Description*
Operid	Specify the user ID to which you want to assign the worklist entry, as a string.

Returns

A Boolean value: true if the reassignment was successful, false otherwise.

Example

```
import PT_WF_WORKLIST:WorklistEntry;

       Local WorklistEntry &wl = create WorklistEntry();
       &wl.busprocname = WF_WORKLIST_VW2.BUSPROCNAME;
       &wl.busactivity = WF_WORKLIST_VW2.ACTIVITYNAME;
       &wl.buseventname = WF_WORKLIST_VW2.EVENTNAME;
       &wl.worklistname = WF_WORKLIST_VW2.WORKLISTNAME;
       &wl.instanceid = WF_WORKLIST_VW2.INSTANCEID;

       If (&wl.SelectByKey()) Then
          If Not (&wl.Reassign("TOPSUSER2")) Then
             /* Reassign error */
          End-If;
       Else
          /* SelectByKey error */
       End-If;
```

See Also

Chapter 27, "Notification Classes," Reassign, page 1467

Save

Syntax

Save()

Description

Use the Save method to save a WorklistEntry to the database.

PeopleCode Event Considerations

You must include this method within events that allow database updates. This includes the following PeopleCode events:

- SavePreChange (Page)
- SavePostChange (Page)
- Workflow
- FieldChange

If this method results in a failure, all database updates are rolled back. All information the user entered into the component is lost, as if the user pressed ESC.

Considerations Using Web Services

This method saves the instance of the worklist to the database. If the worklist entry is marked as worked, (that is, if inststatus property is set to 2) and the worklist entry was originally created by a web service, the originating requestor is sent a response when this method is executed. If the requestor is expecting additional data, use the SaveWithCustomData method instead.

Parameters

None.

Returns

A numeric value: 0 if save didn't complete successfully, nonzero if save completed successfully.

Example

```
If (&worklist.Save() <> 0) Then
   /* success */
Else
   /* handle error */
End-If;
```

See Also

Chapter 27, "Notification Classes," Update, page 1475

Chapter 27, "Notification Classes," SaveWithCustomData, page 1472

Chapter 27, "Notification Classes," inststatus, page 1478

SaveWithCustomData

Syntax

SaveWithCustomData*(&Message,&FieldNameArray*, *&FieldValueArray)*

Description

Use the SaveWithCustomData method to save the worklist entry to the database, as well as to pass additional data back to the requestor that created this worklist entry. You should only use this method with worklist entries that are created by web service. If you want to save a worklist entry without additional data, or a worklist entry that was not created using a web service, use the Save method instead.

If the worklist entry was not created by a web service, or if the worklist entry is not marked as worked (that is, if the inststatus property is not set to 2,) the additional data is ignored.

PeopleCode Event Considerations

You must include this method within events that allow database updates. This includes the following PeopleCode events:

- SavePreChange (Page)
- SavePostChange (Page)
- Workflow
- Message Subscription (the Integration Broker INotificationHandler interface)
- FieldChange

If this method results in a failure, all database updates are rolled back. All information the user entered into the component is lost, as if the user pressed ESC.

Parameters

Parameter	*Description*
&Message	Specify an already instantiated and populated response message containing the custom data to be sent back to the originating web service.
&FieldNameArray	Specify an already instantiated and populated array of string containing the field names to be used by the web service that created the worklist entry.
&FieldValueArray	Specify an already instantiated and populated array of string containing matching field values to be used by the web service that created the worklist entry.

Returns

A message object containing the final message that was sent to the web service if successful, a null object otherwise.

See Also

Chapter 27, "Notification Classes," Save, page 1471

Chapter 27, "Notification Classes," inststatus, page 1478

SelectByKey

Syntax

`SelectByKey``()`

Description

The SelectByKey method uses the key field values that have been assigned to build and execute a Select SQL statement. The field values are then fetched from the database SQL table into the worklist entry and the Select statement is closed.

If you don't specify all the key fields, those you exclude are added to the Where clause with the condition equal to a blank value. If not all keys are set and more than one row is retrieved, you won't receive an error and SelectByKey won't fetch any data.

For worklist entries that were created by web services, use the SelectByMessageId method instead.

Parameters

None.

Returns

A Boolean value: true if the properties are set based on the database values, false otherwise.

See Also

Chapter 27, "Notification Classes," SelectByMessageId, page 1474

SelectByMessageId

Syntax

`SelectByMessageId``()`

Description

Use the SelectByMessageId method to select a worklist entry by the message id. This is useful only for entries that are created by web services.

The following properties must be set before calling this method:

- busactivity
- buseventname
- busprocname
- requestmessageid
- worklistname

Parameters

None.

Returns

A Boolean value: true if this method completes successfully, false otherwise.

See Also

Chapter 27, "Notification Classes," SelectByKey, page 1473

Update

Syntax

`Update``()`

Description

Use the Update method to update the worklist entry with any property values that have changed, and assign values to the associated record in the database. However, this method does not commit the changes to the record in the database. You must use the Save method to update the database.

Parameters

None.

Returns

A numeric value: 0 if update didn't complete successfully, nonzero if update completed successfully.

Example

```
If (&worklist.Update() <> 0) Then
   &worklist.inststatus = "2"  /* mark entry worked */
   /* additional processing */
End-if;
```

See Also

Chapter 27, "Notification Classes," Save, page 1471

WorklistEntry Class Properties

In this section, we discuss the WorklistEntry class properties. The properties are discussed in alphabetical order.

actiondttm

Description

This property returns the Worklist action datetime as a datetime.

This property is read-write.

busactivity

Description

This property returns the source activity name of the source business process as a string.

This property is read-write.

buseventname

Description

This property returns the source event name of the source activity as a string.

This property is read-write.

busprocname

Description

This property returns the source business process name as a string.

This property is read-write.

commentshort

Description

This property returns the short comment the previous user ID gave when reassigning a worklist as a string.

This property is read-write.

do_replicate_flag

Description

This property returns a flag that indicates if the worklist entry must be replicated. Values are:

Value	***Description***
Y	Worklist entry needs to be replicated
N	Worklist entry does not need to be replicated

This property is read-write.

instanceid

Description

This property returns the Instance Id. This property is used with the following properties to make the worklist entry a unique number:

- busprocname
- busactivity
- buseventname
- worklistname

This property is read-write.

instselecteddttm

Description

This property returns the datetime that the worklist was selected to be worked as a datetime value.

This property is read-write.

inststatus

Description

This property returns the Worklist Entry status as a string. The values are:

Value	***Description***
0	new worklist entry
1	selected worklist entry
2	worked worklist entry
3	cancel string

This property is read-write.

insttimeoutddttm

Description

This property returns the datetime the worklist entry timed out. Only some worklist types use this property.

This property is read-write.

instworkeddttm

Description

This property returns the datetime worklist was worked as a datetime value.

This property is read-write.

IsCreatedViaWebService

Description

This property returns a Boolean value: true if the WorklistEntry object was created from a web service, false otherwise.

This property is read-only.

oprid

Description

This property returns the user ID that is assigned the worklist as a string.

This property is read-write.

originatorid

Description

This property returns the user ID that caused the worklist to be created as a string.

This property is read-only.

prevoprid

Description

This property returns the previous user ID who used this worklist entry as a string.

This property is read-write.

requestmessageid

Description

This property is used with worklist entries that are created by web services. It contains the original message ID from the requesting party.

This property is read-write.

ResponseStatus

Description

This property returns the status of the response sent to the originating web service.

Valid values are:

Value	*Description*
0	CreateWorklistEntry message not received yet or this entry was not created by any web service.
1	Message received but not processed yet.
2	Message processed and sent to message queue.
3	Error while sending message to the caller of create worklist entry.
4	Message was sent to the originating web server and was received successfully.

This property is read-write.

syncid

Description

This property is reserved for future use.

This property is read-write.

timedout

Description

This property returns the status flag if the worklist has timed out. Values are:

Value	*Description*
Y	Worklist has timed out
N	Worklist has not timed out

This property is read-write.

transactionid

Description

This property returns the transaction Id as a number, which represents the unit of work. This can be useful for pooled worklists.

This property is read-write.

url

Description

This property returns the URL to work for the worklist as a string. This is used only for replicating worklist entries across PeopleSoft databases.

This property is read-write.

wl_priority

Description

This property returns the priority of the worklist as a string. The values are:

Value	*Description*
low	Worklist has low priority.
medium	Worklist has medium priority.
high	Worklist has high priority.

This property is read-write.

wldaystoselect

Description

This property returns the number of days to select worklist as a number.

This property is read-write.

wldaystowork

Description

This property returns the number of days to work the worklist as a number.

This property is read-write.

worklistname

Description

This property returns the source worklist name for the source event string as a string.

This property is read-write.

WSWorklistEntry Class

This class is used by web services to create Worklist entries.

This class implements the Integration Broker class IRequestHandler.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

WSWorklistEntry Class Methods

The following are the WSWorklistEntry class methods, in alphabetical order.

OnError

Syntax

`OnError`*`(&Message)`*

Description

Use the OnError method to do error handling. This method implements the Integration Broker IRequestHandler interface OnError method.

Parameters

Parameter	***Description***
&Message	Specify an already instantiated and populated message object. This would normally be the message that was used to generate the worklist entry from the web service.

Returns

Depends on implementation.

See Also

Chapter 25, "Message Classes," page 1245

Chapter 39, "SOAPDoc Class," page 2211

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

OnNotify

Syntax

OnNotify*(&Message)*

Description

Use the OnNotify method to creates the worklist entry and the application worklist record.

This method implements the Integration Broker INotificationHandler interface OnNotify method.

Parameters

Parameter	***Description***
&Message	Specify an already instantiated and populated message object. Generally this would be the originating message that created the worklist entry.

Returns

None.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

WSWorklistEntry Class Properties

The following describes the WSWorklistEntry class properties, in alphabetical order.

InstanceID

Description

This property returns the instance ID for the worklist entry, as a string.

This property is read-only.

TransactionID

Description

This property returns the transaction ID for the worklist entry, as a string.

This property is read-only.

Notification Classes Examples

The following are example of typical actions you perform using the notification classes.

See *Enterprise PeopleTools 8.51 PeopleBook: Workflow Technology*, "Using Notification Templates."

Creating a WorklistEntry

The following example creates a WorklistEntry.

The following is the complete code sample: the steps explain each line.


```
import PT_WF_WORKLIST:*;

Local WorklistEntry &worklist;

&worklist = create WorklistEntry();

&worklist.busprocname  = "Administer Workflow";
&worklist.busactivity  = "Send Note";
&worklist.buseventname = "Worklist Note";
&worklist.worklistname = "Worklist Note";

If (&worklist.Create() <> 0) Then
   &worklist.oprid = %UserId;
Else
   /* handle error */
End-If;

If (&worklist.Save() <> 0) Then
   /* success, create a corresponding row in */
   /* application worklist record... */
Else
   /* handle error */
End-If;
```

To create a WorklistEntry:

1. Import the WorklistEntry class.

   You must import the WorklistEntry class before you can use it in your PeopleCode program. In addition, the second line declares the variable &worklist as type WorklistEntry.

   ```
   import PT_WF_WORKLIST:*;

   Local WorklistEntry &worklist;
   ```

2. Instantiate a WorklistEntry object.

   Using the constructor, create an instance of the WorklistEntry class. This does *not* create a WorklistEntry. This creates a WorklistEntry object only. You must use the Create method to actually create the WorklistEntry in the system.

   ```
   &worklist = create WorklistEntry();
   ```

3. Set the required parameters.

   Some of the WorklistEntry class properties are required for every WorklistEntry. You must set a value for these properties before you can update or save the WorklistEntry.

   ```
   &worklist.busprocname  = "Administer Workflow";
   &worklist.busactivity  = "Send Note";
   &worklist.buseventname = "Worklist Note";
   &worklist.worklistname = "Worklist Note";
   ```

4. Create the WorklistEntry.

   After you've set the required values, you need to create the WorklistEntry in the system. This does *not* update the database. This does, however, add values to the notification tables.

   If this method completes successfully, the following properties are populated: instanceid and transactionid.

   - instanceid
   - transactionid

     ```
     If (&worklist.Create() <> 0) Then
        &worklist.oprid = %Userid;
     Else
        /* handle error */
     End-If;
     ```

5. Save the WorklistEntry.

   After you've created the WorklistEntry, save it. The Save method actually updates the database, and so can be used only in events that allow database updates.

   ```
   If (&worklist.Save() <> 0) Then
      /* success, create a corresponding row in */
      /* application worklist record... */
   Else
      /* handle error */
   End-If;
   ```

Updating a WorklistEntry

The following example updates a WorklistEntry so that it is considered worked.

The following is the complete code sample: the steps explain each line.

```
import PT_WF_WORKLIST:*;

Local WorklistEntry &worklist;

&worklist = create WorklistEntry();
&worklist.busprocname = "Administer Workflow";
&worklist.busactivity = "Send Note";
&worklist.buseventname = "Worklist Note";
&worklist.worklistname = "Worklist Note";
&worklist.instanceid = 2;

&ret = &worklist.SelectByKey();

&worklist.inststatus = "2"; /* mark entry worked */
If (&worklist.Update() <> 0) Then
   If (&worklist.Save() <> 0) Then
      /* success */
   Else
      /* handle error */
   End-If;
Else
   /* handle error */
End-If;
```

To update a WorklistEntry:

1. Import the WorklistEntry class.

   You must import the WorklistEntry class before you can use it in your PeopleCode program. In addition, the second line declares the variable &worklist as type WorklistEntry.

   ```
   import PT_WF_WORKLIST:*;
   ```

   ```
   Local WorklistEntry &worklist;
   ```

2. Instantiate a WorklistEntry object.

   Using the constructor, create an instance of the WorklistEntry class. This does *not* create a WorklistEntry. This creates a WorklistEntry object only. You must use the Create method to actually create the WorklistEntry in the database tables, and the Save method to update the tables.

   ```
   &worklist = create WorklistEntry();
   ```

3. Set the required parameters.

   Some of the WorklistEntry class properties are required for every WorklistEntry. You must set a value for these properties before you can update or save the WorklistEntry. The instanceid property, combined with the other properties, indicates a unique WorklistEntry. This value is populated after a WorklistEntry has already been created.

   ```
   &worklist.busprocname  = "Administer Workflow";
   &worklist.busactivity  = "Send Note";
   &worklist.buseventname = "Worklist Note";
   &worklist.worklistname = "Worklist Note";
   &worklist.instanceid = 2;
   ```

4. Populate the worklist entry with data.

   After you've set the keys, populate the worklist entry based on those keys using the SelectByKey method.

   ```
   &ret = &worklist.SelectByKey();
   ```

5. Update and save the WorklistEntry.

   After you've set the required parameters, try to update the WorklistEntry. If that is successful, mark the WorklistEntry as 'worked', and try to save the WorklistEntry. The Save method actually updates the database, and so can be used only in events that allow database updates.

   ```
   If (&worklist.Update() <> 0) Then
      &worklist.inststatus = "2"  /* mark entry worked */
      If (&worklist.Save() <> 0) Then
         /* success */
      Else
         /* handle error */
      End-If;
   Else
         /* handle error */
   End-if;
   ```


Chapter 28

Optimization PeopleCode

This chapter discusses how to:

- Use optimization PeopleCode on the application server.
- Use optimization PeopleCode in an Application Engine program.
- Perform optimization in PeopleCode.
- Use lights-out mode with optimization.
- Use optimization built-in functions.
- Use OptEngine class methods.
- Use OptEngine class properties.
- Use the OptBase application class.
- Use OptBase class methods.
- Use OptInterface class methods.

Important! The optimization PeopleCode classes are not supported on IBM z/OS and Linux for IBM System z platforms.

Using Optimization PeopleCode on the Application Server

While running optimization PeopleCode on the application server, ensure that changed data is committed to the database before calling the CreateOptEngine optimization function and the following OptEngine class methods:

- RunSynch
- RunAsynch
- CheckOptEngineStatus
- ShutDown
- SetTraceLevel
- GetTraceLevel
- InsertOptProbInst

- DeleteOptProbInst

Note. The PeopleCode functions CommitWork and DoSaveNow can be called within a step to save uncommitted data to the database before calling the listed functions and methods. Keep in mind that forcing a commit on pending database updates is a serious step; it prevents roll-back on error. CreateOptEngine, ShutDown, InsertOptProbInst, and DeleteOptProbInst calls modify the database, so take care when terminating the Application Engine program without committing the changes made by those calls.

Using Optimization PeopleCode in an Application Engine Program

When you write an optimization PeopleCode program in an Application Engine program and you schedule it in PeopleSoft Process Scheduler, you must set the process definition with a process type of *Optimization Engine.* Other process types do not allow optimization PeopleCode in Application Engine programs.

While using optimization PeopleCode in Application Engine programs, make sure data is committed before calling the CreateOptEngine optimization function and the following OptEngine class methods:

- RunSynch
- RunAsynch
- CheckOptEngineStatus
- ShutDown
- SetTraceLevel
- GetTraceLevel
- InsertOptProbInst
- DeleteOptProbInst

Note. You can call the PeopleCode functions CommitWork and DoSaveNow within a step to save uncommitted data to the database before calling the listed functions and class methods. Keep in mind that forcing a commit on pending database updates is a serious step; it prevents roll-back on error. CreateOptEngine, ShutDown, InsertOptProbInst, and DeleteOptProbInst calls modify the database, so take care when terminating the Application Engine program without committing the changes made by those calls.

Performing Optimization in PeopleCode

This section discusses how to:

- Create new analytic instances.
- Load analytic instances into an analytic server.
- Run optimization transactions.
- Invoke the Optimization PeopleCode plug-in.

- Shut down optimization engines.
- Delete existing analytic instances.
- Program for database updates.

Creating New Analytic Instances

To create a new analytic instance for an analytic type:

1. Call the function InsertOptProbInst with the analytic type and analytic instance as parameters to create an analytic instance ID.
2. Use Application Engine or a similar mechanism to load the optimization application tables with data.

   Use the analytic instance ID as the key value in scenario-managed optimization application tables.

   The analytic instance is now ready to be loaded into an analytic server.

Note. You can load multiple copies of the same analytic instance into multiple instances of an analytic server, provided that each instance of the analytic server resides in a different application server domain. Each analytic instance loaded into a given domain must be unique. Within a given domain, you cannot have the same analytic instance in more than one analytic server. The analytic server maintains data integrity by checking to see if the data has been altered by another user (refer to the steps in the optimization system architecture description). Try to maintain data consistency when the same analytic instance uses the same database in different domains.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework, "Understanding PeopleSoft Optimization Framework," PeopleSoft Optimization Framework System Architecture

Chapter 28, "Optimization PeopleCode," InsertOptProbInst, page 1515

Loading Analytic Instances Into an Analytic Server

Use the CreateOptEngine function to load an analytic server with an analytic instance. It takes analytic instance ID and a mode parameter with *%Synch* and *%Asynch* as possible values and returns a PeopleCode object of type *OptEngine.*

You can run the PeopleCode on the application server or from Application Engine.

Loading Analytic Instances by Running PeopleCode on the Application Server

To block PeopleCode from running on the application server until the load is done (synchronous mode), use the *%Synch* value for the mode parameter. An error is generated if the load isn't successful. The application server imposes a timeout beyond which the PeopleCode and optimization engine load are terminated. Here is a code example:

```
Local OptEngine &myopt;
&myopt = CreateOptEngine("PATSMITH", %Synch);
```


To load the analytic server without blocking the PeopleCode from running (asynchronous mode) on the application server, use the *%Asynch* value for the mode parameter. The analytic server performs a preliminary check of the load request and returns the OptEngine object if it is successful or an error if it is unsuccessful. A successful return does not mean that the load was successful. You must then use repeated CheckOptEngineStatus methods on the returned OptEngine object to determine whether the analytic engine is done with the load and whether it was successful. Here is a code example:

```
Local OptEngine &myopt;
&myopt = CreateOptEngine("PATSMITH", %Asynch);
```

Loading Analytic Instances by Running PeopleCode in Application Engine

Both synchronous *(%Synch)* and asynchronous *(%Asynch)* modes block the PeopleCode from running on Application Engine until the load is done. Use only *%Asynch* while loading an optimization engine.

The absolute number of optimization engine instances that may be loaded in a given domain is governed by a configuration file loaded by Tuxedo during its domain startup.

See Also

Chapter 28, "Optimization PeopleCode," CheckOptEngineStatus, page 1518

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework, "Administering Optimization Server Components"

Running Optimization Transactions

You send an optimization transaction to the optimization engine using the RunSynch and RunAsynch methods. Both are methods on an OptEngine object. The OptEngine object can be created either by calling CreateOptEngine (if the optimization engine is not loaded already) or by calling GetOptEngine (if the optimization engine is already loaded). Both RunSynch and RunAsynch have the same signature, except that RunSynch runs the optimization transaction in synchronous mode and RunAsynch runs it in asynchronous mode. Both return an integer status code. You can run transactions either on the application server or with Application Engine.

To invoke an optimization transaction:

1. Use the GetOptEngine function to get the OptEngine object as a handle for the optimization engine that has loaded an analytic instance ID.

   Use the CreateOptEngine function to create the OptEngine object for a new optimization engine if the analytic instance has not been loaded.

2. Call RunSynch or RunAsynch to send an optimization transaction to the optimization engine to be run in synchronous or asynchronous mode.

3. If the transaction is run in synchronous mode (RunSynch), use the OptEngine methods GetString, GetNumber, and so on, to retrieve the output result from the optimization transaction.

   The transaction names, parameter names, and data types are viewable in the analytic type in Application Designer.

4. If the transaction is run in asynchronous mode, use the OptEngine method CheckOptEngineStatus to check the status of the optimization transaction in the optimization engine.

   After the transaction is done, result data is available in the database for retrieval using standard PeopleCode mechanisms.

Running Optimization Transactions from the Application Server

To block the PeopleCode from running on the application server until the optimization transaction is done (synchronous mode) and receives the results, use RunSynch to send an optimization transaction. An error status code is returned if the transaction isn't successful. If successful, you can use other methods to retrieve the results from the transaction call. The application server imposes a timeout beyond which the PeopleCode and optimization engine transaction are terminated.

To run a transaction without blocking PeopleCode from running (asynchronous mode) on the application server, use RunAsynch to send an optimization transaction. In this mode, the optimization engine performs a preliminary check of the transaction request and returns a success or failure status code. A successful return does not mean that the transaction is successful; it means only that the syntax is correct. You must then use repeated calls to the CheckOptEngineStatus method on the OptEngine object to determine whether the optimization engine is done with the transaction and whether it is successful.

RunAsynch does not allow transaction output results to be returned. Use this method for long-running transactions that return results entirely through the database.

Running Optimization Transactions from Application Engine

Both synchronous (RunSynch) and asynchronous (RunAsynch) methods block the PeopleCode from running on Application Engine until the optimization transaction is done. RunSynch allows output results to be returned, but it should be used for transactions that are fast (less than 10 seconds). RunAsynch does not have a time limit, but it does not return output results.

See Also

Chapter 28, "Optimization PeopleCode," RunAsynch, page 1534

Invoking the Optimization PeopleCode Plug-In

If you're developing an optimization application that uses the Optimization PeopleCode plug-in, you must do the following to invoke the plug-in:

- Develop a PeopleCode application class that extends the PT_OPT_BASE:OptBase class.
- Define methods in your application class that use the PeopleCode OptInterface class to perform your optimization functions.
- Define an analytic type that specifies the Optimization PeopleCode plug-in, by selecting the PeopleCode Plugin check box in the analytic type properties.
- In the analytic type properties, specify the application package and application class that you developed.
- Define transactions in your analytic type definition that correspond to the methods you developed in your application class, with corresponding parameters.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions," Creating Analytic Type Definitions

Chapter 28, "Optimization PeopleCode," CreateOptInterface, page 1510

Chapter 28, "Optimization PeopleCode," OptBase Application Class, page 1543

Chapter 28, "Optimization PeopleCode," OptInterface Class Methods, page 1565

Shutting Down Optimization Engines

Use the GetOptEngine function to get the OptEngine object as a handle for the optimization engine that loaded an analytic instance ID.

Use the OptEngine method named ShutDown to shut down the optimization engine. This ends the optimization engine process with the current analytic instance ID. Based on application server settings, the system restarts a new, unloaded optimization engine process that can be loaded with any other analytic instance.

See Also

Chapter 28, "Optimization PeopleCode," ShutDown, page 1539

Deleting Existing Analytic Instances

To delete an existing analytic instance for an analytic type:

1. Shut down any optimization engines that have this analytic instance currently loaded.
2. Using Application Engine or a similar mechanism, delete the data in the optimization application tables pertaining to that analytic instance.

   Use the analytic instance ID as the key value to find and delete analytic instance rows from scenario-managed optimization application tables.
3. Use the function DeleteOptProbInst with the analytic type and analytic instance as arguments to delete the analytic instance ID from PeopleTools metadata.

   Note. If you try to delete an existing analytic instance that is loaded in a running optimization engine, DeleteOptProbInst returns *%OptEng_Fail*, and the optional status reference parameter is set to *%OptEng_Exists*.

See Also

Chapter 28, "Optimization PeopleCode," DeleteOptProbInst, page 1511

Programming for Database Updates

You must plan for uncommitted database changes in your optimization PeopleCode. The PeopleSoft Optimization Framework detects pending database updates, and generates a failure status if data is not committed to the database before certain optimization methods are called.

This checking for database updates happens in runtime for the CreateOptEngine function and the following methods: RunSync, RunAsync, Shutdown, GetTraceLevel, and SetTraceLevel. Ensure that your PeopleCode performs proper database updates and commits before you execute these methods. If you use the optional parameter for detailed status that is available for these functions, or the DetailedStatus property that is available for the methods, you can check for the status of %OptEng_DB_Updates_Pending to see if there is a pending database update.

Note. The pending database update may have happened considerably earlier in the code. Forcing a commit within your PeopleCode to avoid this problem prevents roll-back on database error. Forcing a commit should be used with care.

The InsertOptProbInst and DeleteOptProbInst functions can be called only inside FieldChange, PreSaveChange and PostSaveChange PeopleCode events, and in Workflow.

This database update checking happens in compile time for the following functions: InsertOptProbInst and DeleteOptProbInst. Make sure that there are no pending database updates before you execute these methods.

Using Lights-Out Mode with Optimization

This section provides an overview of lights-out mode, and discusses how to:

- Create a request message.
- Create a response message.
- Edit the request PeopleCode.
- Edit the response PeopleCode.

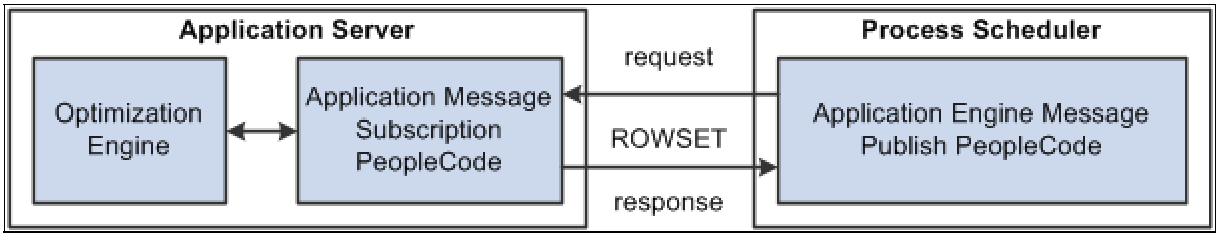
Understanding Lights-out Mode

Some optimization applications can take several hours to run. These are typically run as overnight batch jobs every night when the work load is small to regenerate the optimization solution and have it ready for end users to use in the morning hence the term *lights-out mode.*

In the current release, application messages communicate between the Application Engine batch job and the online optimization engine. After the Application Engine job completes and the optimization solution has been written to the database, an application message initiates the download of the data from the database batch job to the online optimization engine.

Lights-out mode uses an Application Engine PeopleCode program within PeopleSoft Process Scheduler to send requests to an application server and receive responses from it. Within the application server, the OnRequest PeopleCode runs an optimization engine process.

This diagram illustrates the lights-out process:



Lights-out process

This request and response is in the form of a rowset as shown by the example supplied with optimization, the OPT_CALL message. Also supplied as an example is an Application Engine message publish PeopleCode program called PT_OPTCALL.

Important! Application Engine includes an action of type *Log Message.* which PeopleSoft Process Scheduler uses to record its activity in the PS_MESSAGE_LOG table. The PeopleCode MessageBox and WinMessage built-in functions also record their activity in the PS_MESSAGE_LOG table.

During lights-out optimization, these processes can conflict with each other or with the optimization engine when one process locks a row of the table, and another process tries to access the same row.

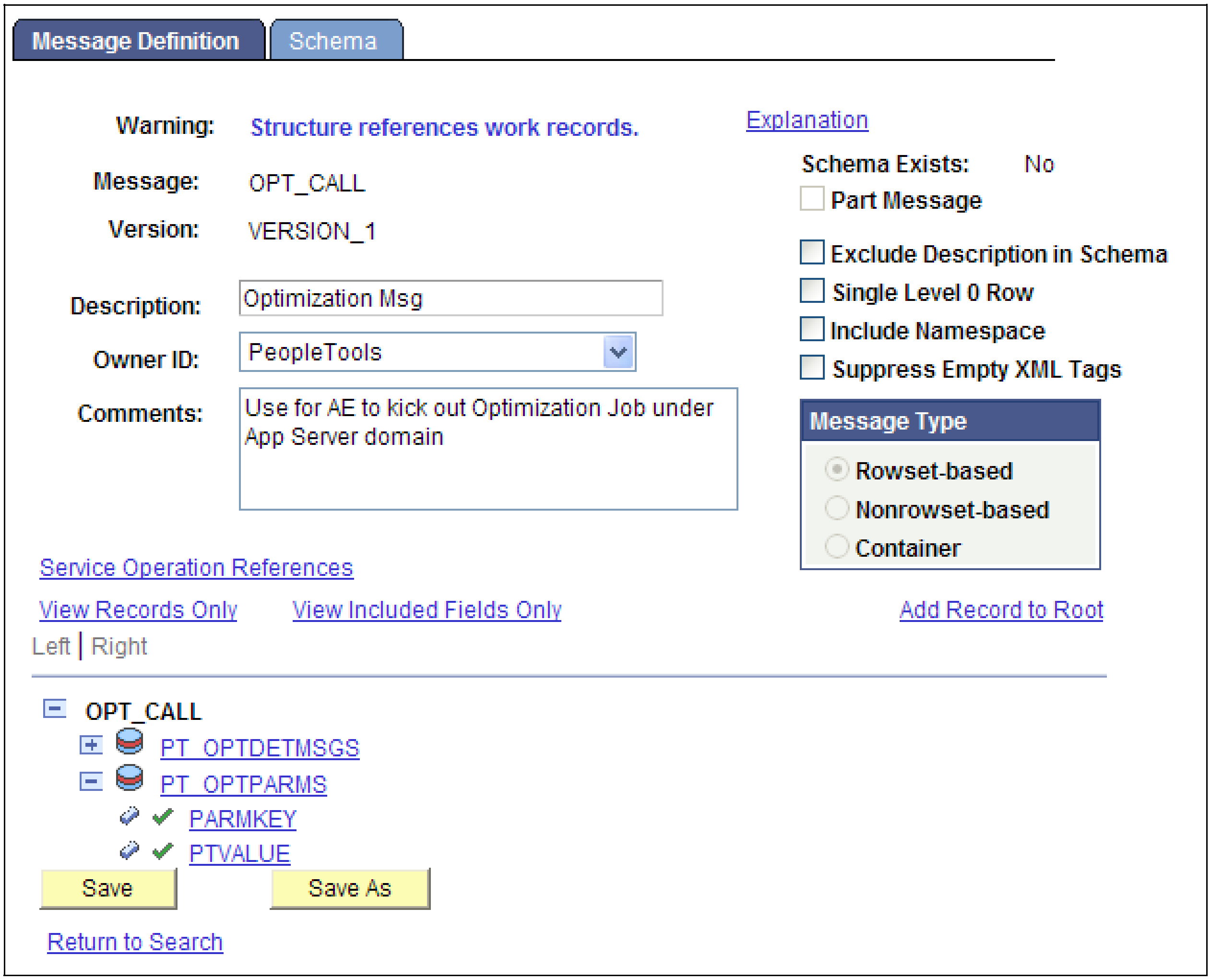
To prevent this conflict, pay close attention to where the MessageBox or WinMessage built-in functions are used in your Application Engine PeopleCode. In general, there can't be any outstanding database updates pending when communicating with the optimization engine using application messages.

The OPT_CALL Message

The OPT_CALL message is an example of what the lights-out process uses as the message for optimization. The OPT_CALL message has a structure using a record, PT_OPTPARMS, having the fields PARMKEY and VALUE which represent a name/value pair. These send requests and responses from the Application Engine PeopleCode in PeopleSoft Process Scheduler to and from the message OnRequest PeopleCode in the application server.

The OPT_CALL message also uses a record, PT_OPTDETMSGS, which contains the information needed for processing a detailed message.

This is an example of the Message Definition page (select PeopleTools, Integration Broker, Integration Setup, Messages) showing the OPT_CALL message definition:



Message Definition page – OPT_CALL message definition

The OPT_CALL message is associated with the OPT_CALL service operation. The OPT_CALL service operation defines the OPT_CALL application package as a handler. This application package implements the Integration Broker methods needed to handle any messaging PeopleCode.

Creating a Request Message

This section provides an overview of the request message and describes how to create messages that:

- Create an optimization engine.
- Check optimization engine status.
- Run an optimization engine transaction.
- Set the trace level.
- Get the trace level.
- Shutdown an optimization engine.

Understanding the Request Message

For optimization, the Application Engine PeopleCode in PeopleSoft Process Scheduler sends a request OPT_CALL message. The message uses rowsets built from PT_OPTPARMS records as the request. You can use the following rowset structures as an example of how to perform certain optimization actions, by sending them as requests from the application engine program in the process scheduler to the message notification PeopleCode in the application server.

Creating an Optimization Engine

To create an optimization engine, structure the rowset as follows, using the PT_OPTPARMS record. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

PARMKEY Field	***VALUE Field***
OPTCMD	CREATE Causes the PeopleCode program implementing the Integration Broker OnRequest method to load an optimization engine. The OPT_CALL example executes the CreateOptEngine function.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.
SYNCH	*Y* if this optimization engine load is to occur synchronously, *N* if asynchronously.

Checking Optimization Engine Status

To check optimization engine status (for example, to see when it finishes loading), structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	***VALUE Field***
OPTCMD	CHECK_STATUS Causes the PeopleCode program implementing the Integration Broker OnRequest method to check the status of an optimization engine. The OPT_CALL example executes the CheckOptEngineStatus function.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.

Running a Transaction

To run a transaction, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	***VALUE Field***
OPTCMD	RUN Causes the PeopleCode program implementing the Integration Broker OnRequest method to run an optimization transaction. The OPT_CALL example executes the GetOptEngine method and either the RunSynch or RunAsynch method.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.
SYNCH	*Y* for a synchronous transaction, *N* for asynchronous.
TRANSACTION	The name of the transaction to run.
The names of one or more transaction parameters.	The value of each named transaction parameter.

Setting the Trace Level

To set a trace level, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	***VALUE Field***
OPTCMD	SET_TRACE_LEVEL Causes the PeopleCode program implementing the OnRequest Integration Broker method to set the severity level at which events will be logged for an optimization engine. The OPT_CALL example executes the SetTraceLevel method.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.
COMPONENT	One of the following values: • %Opt_Engine server activity of the running optimization engine. • %Opt_Utility low level elements that support the running optimization engine. • %Opt_Datacache the in-memory database cache. • %Opt_Plugin the plugin being used for the running optimization engine.

PARMKEY Field	*VALUE Field*
SEVERITY_LEVEL	The severity level to log. The following list starts with the most severe level; the level you specify includes all higher levels. For example, if you specify %Severity_Error, it logs %Severity_Fatal, %Severity_Status, and %Severity_Error messages and filters out the others. • %Severity_Fatal • %Severity_Status • %Severity_Error • %Severity_Warn • %Severity_Info • %Severity_Trace1 • %Severity_Trace2

Getting the Trace Level

To get a trace level, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	*VALUE Field*
OPTCMD	GET_TRACE_LEVEL Causes the PeopleCode program implementing the OnRequest Integration Broker method to get the severity level at which events will be logged for an optimization engine. The OPT_CALL example executes the GetTraceLevel method.
PROBINST	Set to the name of the analytic instance.
PROCINSTANCE	Set to the name of the process instance for this process scheduler job.
COMPONENT	One of the following values: • %Opt_Engine server activity of the running optimization engine. • %Opt_Utility low level elements that support the running optimization engine. • %Opt_Datacache the in-memory database cache. • %Opt_Plugin the plugin being used for the current opt engine.

Shutting Down an Optimization Engine

To shut down an optimization engine, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	SHUTDOWN Causes the PeopleCode program implementing the OnRequest Integration Broker method to shut down an optimization engine. The OPT_CALL example executes the Shutdown method.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.

Creating a Response Message

This section provides an overview of the response message and describes how to create messages that:

- Send optimization status.
- Send a detailed message.

Understanding the Response Message

For optimization, the message PeopleCode in application server receives the request messages, performs an optimization actions, and sends response OPT_CALL messages. One message uses rowsets built from PT_OPTPARMS records, the other uses rowsets from PT_DETMSGS records. You can use the rowset structures in the next section (Sending Optimization Status) as an example of how to send responses from the message notification PeopleCode in the application server to the application engine program in the process scheduler.

Sending Optimization Status

To send the status of the optimization functions and methods called within the PeopleCode program implementing the OnRequest Integration Broker method, structure the rowset as follows using the PT_OPTPARMS record. The optimization functions and messages are called in response to the request input message. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

PARMKEY Field	VALUE Field
STATUS	The return status of the optimization function or method that is called in the message PeopleCode.
DETAILED_STATUS	The optional detailed status returned by many of the optimization functions and methods.

Sending a Detailed Message

To send a detailed message, structure the rowset as follows, using the PT_DETMSGS record. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

PARMKEY Field	***VALUE Field***
MSGSET	The message set number. In the case of optimization, the message set number is 148.
MSGNUM	The name of the detailed message.
PARMCOUNT	The number of message parameters for the detailed message. There can be up to five parameters.
MSGPARM1	The first parameter value.
MSGPARM2	The second parameter value.
MSGPARM3	The third parameter value.
MSGPARM4	The fourth parameter value.
MSGPARM5	The fifth parameter value.

Editing the Request PeopleCode

The PT_OPTCALL Application Engine program serves as a template. It is delivered with all the sections marked as inactive. You can edit the program to suit your needs, then mark the appropriate sections active before running it. You can also use the program as a guide to creating your own Application Engine program.

The program uses these steps to send request messages to perform the following tasks:

1. Load the optimization engine.
2. Wait for the optimization engine load to finish.
3. Run an optimization transaction against the loaded optimization engine.
4. Wait for the optimization transaction to finish running.
5. Set the trace level.
6. Get the trace level.
7. Shut down the optimization engine.

You can edit steps 1 and 3 to run an optimization transaction. You can also use the entire program as a template to create your own Application Engine program.

Loading an Optimization Engine

In step 1, enter the name of your analytic instance. In this example, the name of the analytic instance is *FEMALE1.*

If you have multiple domains, enter the local node name and the machine name and port number for your application server. In this case, the local node name is *%LocalNode* and the machine name and port number are *foo111111:9000.*


```
Local Message &MSG;
Local Message &response;

Component string &probid;
Component string &isSync;
Component string &procinst;
Local integer &nInst;
Local string &url;

Local Rowset &rs;
Local Row &row;
Local Record &rec;

Local string &stName;
Local integer &stVal;

&MSG = CreateMessage(OPERATION.OPT_CALL);
&rs = &MSG.GetRowset();

&row = &rs.GetRow(1);
&rec = &row.GetRecord(Record.PT_OPTPARMS);
&rec.PARMKEY.Value = "OPTCMD";
&rec.VALUE.Value = "CREATE";

&rs.InsertRow(1);
&rec = &rs.GetRow(2).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROBINST";
&rec.VALUE.Value = "FEMALE1";
&probid = "FEMALE1";

&rs.InsertRow(2);
&rec = &rs.GetRow(3).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROCINSTANCE";
&nInst = Record.PT_OPT_AET.PROCESS_INSTANCE.Value;
&rec.VALUE.Value = String(&nInst);
&procinst = String(&nInst);

&rs.InsertRow(3);
&rec = &rs.GetRow(4).PT_OPTPARMS;
&rec.PARMKEY.Value = "SYNCH";
&rec.VALUE.Value = "N";
&isSync = "N";


/* Specify the Application Server domain URL (foo111111:9000 in this example)
*/
&response = %IntBroker.SyncRequest(%LocalNode, "//foo111111:9000 e");

If &response.ResponseStatus = 0 Then
   &stName = &response.GetRowset().GetRow(1).GetRecord(Record.PT_OPTPARMS).Get
Field(Field.PARMKEY).Value;
   &stVal = Value(&response.GetRowset().GetRow(1).GetRecord(Record.PT_
OPTPARMS).GetField(Field.VALUE).Value);
   If &stName = "STATUS" And
         &stVal = %OptEng_Fail Then
      /* Check detailed message here */
      throw CreateException(148, 2, "Can not send to OptEngine");
   End-If;
End-If;
```


Running An Optimization Transaction

In step 3, enter the name of your optimization transaction and its parameter name/value pairs. In this example, the transaction name is *TEST_LONG_TRANS*, the first parameter name/value pair is *Delay_in_Secs* and *30*, and the second parameter name/value pair is *Sleep0_Work1* and *0.*

The parameter values are stored as strings. You may need to convert them in the OnRequest PeopleCode.


```
Local Message &MSG;
Local Message &response;

Local Rowset &rs, &respRS;
Local Row &row;
Local Record &rec, &msgRec;

Component string &probid;
Component string &procinst;
Component string &isSync;
Local string &url = "";
Local integer &parmCount, &msgSet, &msgNum;

&MSG = CreateMessage(OPERATION.OPT_CALL);
&rs = &MSG.GetRowset();

&row = &rs.GetRow(1);
&rec = &row.GetRecord(Record.PT_OPTPARMS);
&rec.PARMKEY.Value = "OPTCMD";
&rec.VALUE.Value = "RUN";

&rs.InsertRow(1);
&rec = &rs.GetRow(2).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROBINST";
&rec.VALUE.Value = &probid;

&rs.InsertRow(2);
&rec = &rs.GetRow(3).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROCINSTANCE";
&rec.VALUE.Value = &procinst;

&rs.InsertRow(3);
&rec = &rs.GetRow(4).PT_OPTPARMS;
&rec.PARMKEY.Value = "SYNCH";
&rec.VALUE.Value = &isSync;

&rs.InsertRow(4);
&rec = &rs.GetRow(5).PT_OPTPARMS;
&rec.PARMKEY.Value = "TRANSACTION";
&rec.VALUE.Value = "TEST_LONG_TRANS";

&rs.InsertRow(5);
&rec = &rs.GetRow(6).PT_OPTPARMS;
&rec.PARMKEY.Value = "Delay_in_Secs";
&rec.VALUE.Value = "30";

&rs.InsertRow(6);
&rec = &rs.GetRow(7).PT_OPTPARMS;
&rec.PARMKEY.Value = "Sleep0_Work1";
&rec.VALUE.Value = "0";

/* SyncRequest will carry a url */
SQLExec("select URL from PSOPTSTATUS where PROBINST=:1 AND URL NOT LIKE '%:0';",
 &probid, &url);
If &url = "" Then
   throw CreateException(148, 2, "Can not send to OptEngine");
End-If;

/* Specify the Application Server domain URL.
   (This was specified in Step 1 in this example.)
*/
&response = %IntBroker.SyncRequest(%LocalNode, &url);

If &response.ResponseStatus = 0 Then
```



```
   &stName = &response.GetRowset().GetRow(1).GetRecord(Record.PT_OPTPARMS).Get
Field(Field.PARMKEY).Value;
   &stVal = Value(&response.GetRowset().GetRow(1).GetRecord(Record.PT_
OPTPARMS).GetField(Field.VALUE).Value);

   If &stName = "STATUS" And
         &stVal = %OptEng_Fail Then
      throw CreateException(148, 2, "Can not send to OptEngine");
   End-If;

   /* Check Detailed msg here */
   If &isSync = "Y" And
         &stVal = %OptEng_Success Then

      &respRS = &response.GetRowset();
      &rowNum = &respRS.ActiveRowCount;
      For &iloop = 1 To &rowNum
         &msgRec = &respRS.GetRow(&iloop).GetRecord(Record.PT_OPTDETMSGS);
         If (&msgRec.GetField(Field.MSGSET).Value <> 0) Then
            &msgSet = Value(&msgRec.GetField(Field.MSGSET).Value);
            &msgNum = Value(&msgRec.GetField(Field.MSGNUM).Value);
            &parm1 = &msgRec.GetField(Field.MSGPARM1).Value;
            &parm2 = &msgRec.GetField(Field.MSGPARM2).Value;
            &parm3 = &msgRec.GetField(Field.MSGPARM3).Value;
            &parm4 = &msgRec.GetField(Field.MSGPARM4).Value;
            &parm5 = &msgRec.GetField(Field.MSGPARM5).Value;
            &string = MsgGetText(&msgSet, &msgNum, "Message Not Found", &parm1,
 &parm2, &parm3, &parm4, &parm5);

         End-If;
      End-For;

   End-If;

End-If;
```

Editing the Response PeopleCode

The OPT_CALL message definition serves as a template. It is delivered to work with the PT_OPTCALL Application Engine program. You can edit the program to suit your needs, or use it as a guide when creating your own response message program.

OPT_CALL Message Program

The OPT_CALL application package implements the Integration Broker method OnRequest. The PeopleCode in this method shows application messages for lights-out mode.

Depending upon the request message, the OnRequest method PeopleCode calls appropriate optimization functions and methods to perform these tasks, and sends a response message containing the returned status and detailed messages from the optimization functions and methods.

You can use the OnRequest method PeopleCode as a template to create your own response message PeopleCode program. For example, you can edit it to run an optimization transaction, which is shown below as an example. This example is edited to match the examples for step 1 and step 3 in the PT_OPTCALL program.

Processing the Transaction Parameters

Edit the OPT_CALL application program OnRequest method to enter the name of your optimization transaction and the name/value pairs for its parameters. In this example, the transaction name is *TEST_LONG_TRANS,* the first parameter name/value pair is *&delayParm* and *&delay* (maps to *Delay_in_Secs* from the request message), and the second parameter name/value pair is *&sleepParm* and *&isSleep* (maps to *Sleep0_Work1* from the request message).

The parameter values are stored as strings in step 3 of the Application Engine program. You may need to convert them here to your desired format. Here is a section of the application program showing the places to edit.

```
If &trans = "TEST_LONG_TRANS" Then
   &REC = &rs.GetRow(6).PT_OPTPARMS;
   &delayParm = &REC.PARMKEY.Value;
   &delay = Value(&REC.VALUE.Value);

   &REC = &rs.GetRow(7).PT_OPTPARMS;
   &sleepParm = &REC.PARMKEY.Value;
   &isSleep = Value(&REC.VALUE.Value);

   &myopt = GetOptEngine(&inst, &detStatus);
   If (&myopt = Null) Then
      &optstatus = %OptEng_Fail;
   End-If;

   If &myopt <> Null And &isSync = "Y" Then
      &optstatus = &myopt.RunSynch(&trans, &delayParm, &delay, &sleepParm, &isSleep
);
      &detStatus = &myopt.DetailedStatus;
      End-If;

   If &myopt <> Null And &isSync = "N" Then
      &myopt.ProcessInstance = &procInst;
      &optstatus = &myopt.RunASynch(&trans, &delayParm, &delay, &sleepParm, &is
Sleep);
      &detStatus = &myopt.DetailedStatus;
   End-If; /* iif myopt=null */
End-If;
```

Building a Status Response Message

This section shows the a response message to send a status message for the OPT_CALL message in the application server.

```
/* Insert detailed status and detailed msgs into Response msg rowset */
&respRS = &response.GetRowset();
&respRS.GetRow(1).GetRecord(Record.PT_OPTPARMS).GetField(Field.PARMKEY).Value =
 "STATUS";
&respRS.GetRow(1).GetRecord(Record.PT_OPTPARMS).GetField(Field.VALUE).Value =
 String(&optstatus);

&respRS.InsertRow(1);
&respRS.GetRow(2).GetRecord(Record.PT_OPTPARMS).GetField(Field.PARMKEY).Value =
 "DETAILED_STATUS";
&respRS.GetRow(2).GetRecord(Record.PT_OPTPARMS).GetField(Field.VALUE).Value =
 String(&detStatus);
```


Building a Detailed Response Message

This section shows a response message to send a detailed message for the OPT_CALL message on the application server.

```
/*Either optcmd or inst is not passed in correctly, or optengine is not loaded
/created correctly */
If &myopt = Null Then
   &msgRec = &respRS.GetRow(1).GetRecord(Record.PT_OPTDETMSGS);
   If &isParmBad = True Then
      &msgRec.GetField(Field.MSGSET).Value = 148;
      &msgRec.GetField(Field.MSGNUM).Value = 505;
   End-If;
End-If;

/* If it is sync transaction, insert DetailMsg to response msg */
If &myopt <> Null And
      &isSync = "Y" And
      &optcmd = "RUN" And
      &optstatus = %OptEng_Success Then
   &arrArray = &myopt.DetailMsgs;
   For &iloop = 1 To &arrArray.Len
      /* First two rows have been inserted because of PT_OPTPARMS for two status
 codes */
      If &iloop > 2 Then
         &respRS.InsertRow(&iloop - 1);
      End-If;

      &msgRec = &respRS.GetRow(&iloop).GetRecord(Record.PT_OPTDETMSGS);
      &msgRec.GetField(Field.MSGSET).Value = String(&arrArray [&iloop][1]);
      &msgRec.GetField(Field.MSGNUM).Value = String(&arrArray [&iloop][2]);
      &msgRec.GetField(Field.PARMCOUNT).Value = String(&arrArray [&iloop][3]);
      &msgRec.GetField(Field.MSGPARM1).Value = String(&arrArray [&iloop][4]);
      &msgRec.GetField(Field.MSGPARM2).Value = String(&arrArray [&iloop][5]);
      &msgRec.GetField(Field.MSGPARM3).Value = String(&arrArray [&iloop][6]);
      &msgRec.GetField(Field.MSGPARM4).Value = String(&arrArray [&iloop][7]);
      &msgRec.GetField(Field.MSGPARM5).Value = String(&arrArray [&iloop][8]);
   End-For;

End-If;
```

Optimization Built-in Functions

This section discusses the optimization functions. The functions are discussed in alphabetical order.

CreateOptEngine

Syntax

CreateOptEngine(*analytic_inst,* {%Synch | %ASynch}[, *&detailedstatus*] [, *processinstance*])

Description

The CreateOptEngine function instantiates an OptEngine object, loads an optimization engine with an analytic instance and returns a reference to it.

Parameters

Parameter	***Description***
Analytic_inst	Specify the analytic instance ID, which is a unique ID for this analytic instance in this optimization engine. This is supplied by users when they request that an optimization be run.
%Synch | %Asynch	Specify whether the optimization engine is synchronous or asynchronous. The values are: • %Synch: run the optimization engine synchronously. • %Asynch: run the optimization engine asynchronously.
&detailedstatus	Specify a variable that the engine uses to give further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Aiid: The analytic instance ID passed to the function is invalid. • %OptEng_Exists: An optimization engine instance already exists and is loaded. • %OptEng_Method_Disabled: A method is disabled or not valid. • %OptEng_DB_Updates_Pending: indicates that database updates are pending.
processinstance	Enter the process instance ID. You use this parameter only with lights-out processing, most likely with the subscription PeopleCode for application message. **Note.** This optional parameter is positional. If you use it, you must also use the *&detailedstatus* parameter. The state record that you use with Application Engine contains the process instance ID. See Chapter 28, "Optimization PeopleCode," Using Lights-Out Mode with Optimization, page 1495. See *Enterprise PeopleTools 8.51 PeopleBook: Application Engine*, "Developing Efficient Programs," Using State Records.

Returns

If successful, CreateOptEngine returns an OptEngine PeopleCode object. If the function fails, it returns a null value. Examine the optional status reference parameter in case of a Null return for additional information regarding the failure.

Example

An OptEngine object variable can be scoped as Local, Component, or Global.

You declare OptEngine objects as type OptEngine. For example:

```
Local OptEngine &MyOptEngine;
Component OptEngine &MyOpt;
Global OptEngine &MyOptEng;
```

The following example loads an optimization engine with the analytic instance:

```
Local OptEngine &myopt;
Local string &probinst;
Local string &transaction;
Local integer &detailedstatus;

&probinst = GetRecord(Record.PSOPTPRBINST).GetField(Field.PROBINST).Value;
&myopt = CreateOptEngine(&probinst, %Synch);
```

The following example shows the use of the optional status parameter:

```
&myopt = CreateOptEngine(&probinst, %Synch, &detailedstatus);
if &myopt = Null then
    if &detailedstatus = %OptEng_Invalid_Piid then
        /*perform some action */
    end_if;
end_if;
```

CreateOptInterface

Syntax

CreateOptInterface()

Description

The CreateOptInterface function instantiates an OptInterface object.

Note. You can use this function and the OptInterface methods only within an application class that you extend from the OptBase application class, or within PeopleCode that you call from that application class. This ensures that the OptInterface PeopleCode runs only on the optimization engine.

Parameters

None.

Returns

If successful, CreateOptInterface returns an OptInterface PeopleCode object. If the function fails, it returns a null value.

Example

You declare OptInterface objects as type OptInterface. For example:

```
Local OptInterface &MyOptInterface;
Component OptInterface &MyOptInt;
Global OptInterface &MyOptInt;
```

The following example instantiates an OptInterface object:

```
Local OptInterface &myInterface;
Int &status;

&myInterface = CreateOptInterface(&addtionalStatus);
if (&myInterface != NULL) then
    &status = &myInterface.ActivateModel("RMO_TEST");
        if (&status = %OptInter_Fail) then
            /* examine &myInterface.DetailedStatus for reason */
            ...
        end-if;
else
    /* CreateOptInterface has returned NULL */
    /* take some corrective action here */
        ...
end_if;
```

DeleteOptProbInst

Syntax

DeleteOptProbInst(*probinst*[, *&detailedstatus*])

Description

The DeleteOptProbInst function deletes the analytic instance ID from PeopleTools metadata. This function can be called only inside FieldChange, PreSaveChange and PostSaveChange PeopleCode events, and in Workflow.

Note. Use this function to delete the analytic instance ID after deleting data in optimization application tables for this analytic instance.

Parameters

Parameter	*Description*
probinst	Enter the analytic instance ID to delete.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The analytic instance ID passed to the function is invalid. • %OptEng_Sql_Exception: A SQLerror is encountered when access database. • %OptEng_Exists: An analytic server loaded with this analytic instance still exists.

Returns

Returns %OptEng_Success if successful; otherwise returns %OptEng_Fail.

Example

The following example deletes the instance for an analytic type:

Note. Whenever you add records to an analytic type, you must call DeleteOptProbInst to delete the old analytic type instances and then call InsertOptProbInst to recreate them.

```
Local string &probinst;
Local string &probtype;
Local integer &ret;
&probinst = "PATSMITH";
&probtype = "QEOPT";
&ret = DeleteOptProbInst(&probinst, &probtype);
If &ret <> %OptEng_Success  Then
   QEOPT_WRK.MESSAGE_TEXT = "Delete of analytic instance " | &probinst | "
 failed.";
Else
   QEOPT_WRK.MESSAGE_TEXT = "Analytic Instance " | &probinst | " deleted.";
End-If;
```

The following example shows the use of the optional status parameter:


```
Local integer &detailedstatus;
&ret = DeleteOptProbInst(&probinst, &probtype, &detailedstatus);
If &ret <> %OptEng_Success  AND &detailedstatus=%OptEng_Invalid_Piid then
   QEOPT_WRK.MESSAGE_TEXT = "Delete of analytic instance " | &probinst | " failed
for bad piid.";
Else
   QEOPT_WRK.MESSAGE_TEXT = "Analytic Instance " | &probinst | deleted.";
End-If;
```

GetOptEngine

Syntax

GetOptEngine(*probinst*[,*&detailedstatus*])

Description

The GetOptEngine function returns a handle to an optimization engine that is already loaded with the analytic instance.

Note. You cannot call GetOptEngine from a domain other than the application server.

Parameters

Parameter	*Description*
probinst	Enter the analytic instance ID, which is unique ID for this analytic instance in this optimization engine.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The analytic instance ID passed to the function is invalid.

Returns

Returns an OptEngine PeopleCode object if successful, a null value otherwise.

Example

The following example causes an optimization engine to shut down its analytic instance:


```
Global string &probinst;
Local OptEngine &myopt;
Local integer &status;

&myopt = GetOptEngine(&probinst);
If &myopt <> NULL then
&status = &myopt.ShutDown();
QEOPT_WRK.MESSAGE_TEXT = "Analytic Instance ID " | &probinst
 | " has been shutdown successfully.";
End-if;
```

Or, you can use the optional status parameter:

```
&myopt = GetOptEngine(&probinst, &detailedstatus);
if &myopt=NULL and &detailedstatus=%OptEng_Invalid_Piid then
     /* perform some action */
End-if;
```

GetOptProbInstList

Syntax

GetOptProbInstList(*ProblemType* , *OutputErrorCode* [, *Prefix*] [, *&detailedstatus*])

Description

The GetOptProbInstList function gets the list of all analytic instance IDs in an analytic type.

Parameters

Parameter	*Description*
ProblemType	Enter the name of the analytic type that you created in Application Designer.
OutputErrorCode	Future use. Always returns zero.
Prefix	(Optional) Enter a string. If used, this prefix causes the returned list to include only the analytic instance IDs that start with this prefix. If not used, all the analytic instance IDs in the analytic type are returned.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The analytic type name passed to the function is invalid.

Returns

Returns an array of strings containing the optimization analytic instance list.

Example

The following example shows the usage of GetOptProbInstList to fill the display field on a page:

```
Global string &probinst;
Local integer &detailedstatus;
Local integer &iloop;
Local array of string &instarray;

QEOPT.OPERATOR = %UserId;

&instarray = GetOptProbInstList(QEOPT.PROBTYPE, &ret, &detailedstatus);

If &ret <> %OptEng_Success Then
   QEOPT_WRK.MESSAGE_TEXT = "Could not get analytic instances
 for analytic type " | QEOPT.PROBTYPE ;
Else
   For &iloop = 1 To &instarray.Len
      QEOPT_WRK.MESSAGE_TEXT = QEOPT_WRK.MESSAGE_TEXT | &instarray[&iloop] | " ";
   End-For;
End-If;
```

The following example shows the use of the optional status parameter:

```
&instarray = GetOptProbInstList(QEOPT.PROBTYPE, &ret, &detailedstatus);
If &ret <> %OptEng_Success and &detailedstatus=%OptEng_Invalid_Piid Then
   QEOPT_WRK.MESSAGE_TEXT = "Could not get analytic instances for analytic type "
| QEOPT.PROBTYPE | "because bad piid" ;
End-If;
```

InsertOptProbInst

Syntax

InsertOptProbInst(*probinst, ProblemType*[, *&detailedstatus*] [,*Description*])

Description

The InsertOptProbInst function inserts a new analytic instance ID into the PeopleTools metadata.

The InsertOptProbInst function can be called only inside FieldChange, PreSave and PostSave PeopleCode events, and in Workflow.

Note. You must use this function to create the analytic instance ID before inserting data into optimization application tables for this analytic instance.

Parameters

Parameter	*Description*
probinst	Enter the analytic instance ID to be inserted into the analytic type.
ProblemType	Enter the name of the analytic type that you created in Application Designer.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The analytic instance ID passed to the function is invalid.
Description	(Optional) Specify a description for the analytic instance. This parameter takes a string value.

Returns

This method returns a constant. Valid values are:

Value	*Description*
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

```
Local string &probinst;
Local string &probtype;
Local integer &ret;
Local integer &detailedstatus;

&probinst = "PATSMITH";
&probtype = "QEOPT";
&probDescr = "New QEOPT instance";
&ret = InsertOptProbInst(&probinst, &probtype, &probDescr);
If &ret <> %OptEng_Success Then
   QEOPT_WRK.MESSAGE_TEXT = "Insert of analytic instance "
 | &probinst | " failed.";
Else
   QEOPT_WRK.MESSAGE_TEXT = "Analytic Instance " | &probinst | " created.";
End-If;
```

The following example shows the use of the optional status parameter:


```
&ret = InsertOptProbInst(&probinst, &probtype, &detailedstatus);
If &ret <> %OptEng_Success and &detailedstatus=%OptEng_Invalid_Piid Then
   QEOPT_WRK.MESSAGE_TEXT = "Insert of analytic instance "
 | &probinst | " failed for bad piid.";
End-if;
```

IsValidOptProbInst

Syntax

IsValidOptProbInst*(probinst* [, &*detailedstatus*])

Description

IsValidOptProbInst determines if a given analytic instance exists in the optimization metadata.

Parameters

Parameter	***Description***
probinst	Enter the analytic instance ID to be validated.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Invalid_Piid: The analytic type name passed to the function is invalid.

Returns

This method returns a constant. Valid values are:

Value	***Description***
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

```
Local string &probinst;
Local integer &detailedstatus;
Local integer &ret;

&probinst = "PATSMITH";
&ret = IsValidOptProbInst(&probinst, &detailedstatus);
If &ret <> %OptEng_Success and &detailedstatus=%OptEng_Invalid_Piid Then
   <perform some action>
End-if;
```

OptEngine Class Methods

This section discusses the optimization methods for the OptEngine PeopleCode class. The methods are listed in alphabetical order.

CheckOptEngineStatus

Syntax

CheckOptEngineStatus()

Description

The CheckOptEngineStatus method returns the status of the optimization engine, using a combination of its return value and the DetailedStatus OptEngine class property. Keep the following in mind:

- The value returned by CheckOptEngineStatus is the operational status of the optimization engine.
- The DetailedStatus property indicates the completion status of the OptEngine method call CheckOptEngineStatus.

For example, CheckOptEngineStatus can return %OptEng_Idle and DetailedStatus is %OptEng_Success. For CheckOptEngineStatus, DetailedStatus can have the value:

- %OptEng_Success
- %OptEng_Fail
- %OptEng_Not_Available

Note. Before this method is called, the CreateOptEngine or GetOptEngine must be called.

Returns

Returns an integer for the status of the optimization engine. These numbers are message IDs belonging to message set 148 in the message catalog.

Numeric Value	***Constant Value***	***Description***
21	%OptEng_Not_Loaded	The optimization engine process is running, but is not currently loaded with an application problem.
22	%OptEng_Busy_Loading	The optimization engine is busy loading an application problem. It will not accept transaction requests until loading completes.
23	%OptEng_Idle	The optimization engine is loaded with an application problem and waiting for a transaction request.
24	%OptEng_Busy	The optimization engine is busy processing a transaction request for the loaded application problem. It will not accept additional transaction requests until the current one completes.
26	%OptEng_Unknown	An error has occurred. The optimization engine status cannot be determined.

Example

This PeopleCode example shows optimization engine status being checked:

```
Local OptEngine &myopt;
Local string &probinst;
Local integer &status;
&myopt = GetOptEngine("PATSMITH");
/* Initialize the DESCRLONG field in the QE_FUNCLIB_OPT record to null.  */
GetLevel0().GetRow(1).GetRecord(Record.QE_FUNCLIB_OPT).DESCRLONG.Value = "";
&status = &myopt.CheckOptEngineStatus();
GetLevel0().GetRow(1).GetRecord(Record.QE_FUNCLIB_OPT).DESCRLONG.Value = "Opt
Engine status = " | MsgGet(148, &status, "Could not send to the OptEngine.");
```

You can also retrieve the detailed status:

```
Local integer &detailedstatus
&status = &myopt.CheckOptEngineStatus();
&detailedstatus = &myopt.DetailedStatus;
```


FillRowset

Syntax

`FillRowset``(`*`PARAM_NAME, &Rowset`*`[, `*`&functionstatus`*`])`

Description

This method gets the value of a transaction output parameter that is a rowset. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

When using the OptEngine DetailedStatus property, keep the following in mind:

- The value returned by FillRowset is the operational status of the optimization engine.
- The OptEngine DetailedStatus property indicates the completion status of the OptEngine method call FillRowset.

For example, FillRowset returns %OptEng_Fail, and DetailedStatus is %OptEng_Method_Disabled.

For FillRowset, the DetailedStatus property can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled.

  This indicates that the method is disabled or not valid.
- %OptEng_Wrong_Parm_Type

Parameters

Parameter	***Description***
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.
&Rowset	Enter the rowset containing the values. This rowset must be a single record rowset, and the record must match the record name associated with the transaction parameter in the analytic type definition.

Parameter	***Description***
&functionstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • OptEng_Success: The function completed successfully. • OptEng_Fail: The function failed. • OptEng_Method_Disabled: A method is disabled or not valid.

Returns

This method returns a constant. Valid values are:

Value	***Description***
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

The following PeopleCode example runs a synchronous optimization transaction named RETURN_MACHINE_UNAVAILABLE. It has these parameters:

- Input: MACHINE_NAME to specify the machine for which we need unavailable times.
- Output: RETURN_TIMES to specify a rowset and MACHINE_WRK record containing the BEGIN_DATE and END_DATE fields.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local Rowset &rs;
&myopt = GetOptEngine("PATSMITH");
&machname = QEOPT_WRK.MACHINE_NAME.Value;
/* Run the RETURN_MACHINE_UNAVAILABLE transaction synchronously with input values.
  */
&status = &myopt.RunSynch("RETURN_MACHINE_UNAVAILABLE", "MACHINE_NAME", &machname);
If Not &status Then
   QEOPT_WRK.MESSAGE_TEXT = " RETURN_MACHINE_UNAVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the RETURN_MACHINE_UNAVAILABLE transaction.  */
&rs = CreateRowset(Record.MACHINE_WRK);
&status = &myopt.FillRowset("RETURN_TIMES", &rs);
```

You can also use the [new->] DetailedStatus property as follows:


```
&status = &myopt.FillRowset("RETURN_TIMES", &rs);
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
    /* perform some action */
End-if;
```

GetDate

Syntax

GetDate(*PARAM_NAME*[, *&status*])

Description

This method gets the value of a transaction output parameter with a data type of Date. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The OptEngine DetailedStatus property indicates the completion status of the OptEngine method call GetDate. For GetDate, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	***Description***
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a Date object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 28, "Optimization PeopleCode," GetNumber, page 1526.

GetDateArray

Syntax

GetDateArray*(PARAM_NAME)*

Description

This method gets the value of a transaction output parameter with a data type Array of Date. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The OptEngine DetailedStatus property indicates the completion status of the OptEngine method call GetDateArray. For GetDateArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	***Description***
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns an Array of Date object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 28, "Optimization PeopleCode," GetStringArray, page 1529.

GetDateTime

Syntax

`GetDateTime`*`(PARAM_NAME)`*

Description

This method gets the value of a transaction output parameter with a data type of DateTime. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetDateTime. For GetDateTime, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	***Description***
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a DateTime object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 28, "Optimization PeopleCode," GetNumber, page 1526.

GetDateTimeArray

Syntax

```
GetDateTimeArray(PARAM_NAME)
```

Description

This method gets the value of a transaction output parameter with a data type Array of DateTime. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetDateTimeArray. For GetDateTimeArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	***Description***
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns an Array of DateTime object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 28, "Optimization PeopleCode," GetStringArray, page 1529.

GetNumber

Syntax

GetNumber*(PARAM_NAME)*

Description

This method gets the value of a transaction output parameter with a data type of Number. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetNumber. For GetNumber, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	***Description***
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a Number object; use this method when that is the data type of the transaction output parameter value.

Example

The following PeopleCode example runs a synchronous optimization transaction named IS_MACHINE_AVAILABLE. It has these parameters:

- Input MACHINE_NAME to specify the machine.
- Inputs BEGIN_DATE and END_DATE to specify the time slot.
- Output AVAILABLE_FLAG to specify whether the machine is available in that time slot.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local datetime &begindate;
Local datetime &enddate;
&myopt = GetOptEngine("PATSMITH");
&machname = QEOPT_WRK.MACHINE_NAME.Value;
&begindate = QEOPT_WRK.BEGIN_DATE.Value;
&enddate = QEOPT_WRK.END_DATE.Value;
/* Run the IS_MACHINE_AVAILABLE transaction synchronously with input values.  */
&status = &myopt.RunSynch("IS_MACHINE_AVAILABLE", "MACHINE_NAME",
 &machname, "BEGIN_DATE", &begindate, "END_DATE", &enddate);
If Not &status Then
   QEOPT_WRK.MESSAGE_TEXT = "IS_MACHINE_AVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the IS_MACHINE_AVAILABLE transaction.  */
QEOPT_WRK.AVAILABLE_FLAG = &myopt.GetNumber("AVAILABLE_FLAG");
```

You can use the DetailedStatus property as follows:

```
QEOPT_WRK.AVAILABLE_FLAG = &myopt.GetNumber("AVAILABLE_FLAG");
if &myopt.DetailedStatus=%OptEng_Fail then
   /* perform some action */
End-if;
```

GetNumberArray

Syntax

GetNumberArray*(PARAM_NAME)*

Description

This method gets the value of a transaction output parameter with a data type Array of Number. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetNumberArray. For GetNumberArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: this indicates that the method is disabled or not valid.

Note. Do not pass an array of type Integer as a transaction parameter. Use an array of type Number instead.

Parameters

Parameter	***Description***
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns an Array of Number object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 28, "Optimization PeopleCode," GetStringArray, page 1529.

GetString

Syntax

`GetString`*`(PARAM_NAME)`*

Description

This method gets the value of a transaction output parameter with a data type of String. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetString. For GetString, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	*Description*
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a String object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 28, "Optimization PeopleCode," GetNumber, page 1526.

GetStringArray

Syntax

`GetStringArray`*`(PARAM_NAME)`*

Description

This method gets the value of a transaction output parameter with a data type Array of String. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetStringArray. For GetStringArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns an Array of String object; use this method when that is the data type of the transaction output parameter value.

Example

The following PeopleCode example runs a synchronous optimization transaction named ARE_MACHINES_AVAILABLE. It has these parameters:

- Inputs BEGIN_DATE and END_DATE to specify the time slot.
- Output MACHINE_NAMES to specify the machines available in that time slot.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local array of string &machnames;
Local datetime &begindate;
Local datetime &enddate;
&myopt = GetOptEngine("PATSMITH");
&begindate = QEOPT_WRK.BEGIN_DATE.Value;
&enddate = QEOPT_WRK.END_DATE.Value;
/* Run the ARE_MACHINES_AVAILABLE transaction synchronously with input values.  */
&status = &myopt.RunSynch("ARE_MACHINES_AVAILABLE",
 "BEGIN_DATE", &begindate, "END_DATE", &enddate);
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "ARE_MACHINES_AVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the ARE_MACHINES_AVAILABLE transaction.  */
&machnames = &myopt.GetStringArray("MACHINE_NAMES");
```

The following example shows the use of the DetailedStatus property:

```
Local array of string &machnames;
&machnames = &myopt.GetStringArray("MACHINE_NAMES");
if &myopt.DetailedStatus=%OptEng_Fail then
   /* perform some action */
End-if;
```

GetTime

Syntax

GetTime*(PARAM_NAME)*

Description

This method gets the value of a transaction output parameter with a data type of Time. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetTime. For GetTime, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	***Description***
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a Time object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 28, "Optimization PeopleCode," GetNumber, page 1526.

GetTimeArray

Syntax

`GetTimeArray`*`(PARAM_NAME)`*

Description

This method gets the value of a transaction output parameter with a data type Array of Time. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetTimeArray. For GetTimeArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	*Description*
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns an Array of Time object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 28, "Optimization PeopleCode," GetStringArray, page 1529.

GetTraceLevel

Syntax

`GetTraceLevel(component)`

Description

GetTraceLevel gets the severity level at which events are logged for a given component.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetTraceLevel. For GetTraceLevel, DetailedStatus can have the value:

- %OptEng_Success.

  This indicates that the function completed successfully.

- %OptEng_Fail.

  This indicates that the function failed.

- %OptEng_Method_Disabled.

  This indicates that the method is disabled or not valid.

- %OptEng_DB_Updates_Pending.

  This indicates that database updates are pending.

Parameters

Parameter	***Description***
component	Enter one of the following PeopleCode constants: Opt_Engine, Opt_Utility, Opt_Datacache, or Opt_Plugin.

Returns

Returns one of the following.

- %Severity_Fatal
- %Severity_Status
- %Severity_Error
- %Severity_Warn
- %Severity_Info

- %Severity_Trace1
- %Severity_Trace2

Example

```
Local OptEngine &myopt;
Local integer &tracelevel;

&myopt = GetOptEngine("PATSMITH");

&tracelevel = &myopt.GetTraceLevel(%Opt_Engine);
if &myopt.DetailedStatus = %OptEng_Success then

   if (&tracelevel = %Severity_Info_ then
      winmessage("Severity level for the OptEngine is 'Info'");
   End-if;
End-if;
```

RunAsynch

Syntax

RunAsynch*(TRANSACTION, PARM_PAIRS)*

Description

The RunAsynch method requests the optimization engine to run the transaction in asynchronous mode.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by RunASynch is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call RunASynch.

For example, RunASynch can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For RunASynch, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Parameters

Parameter	*Description*
TRANSACTION	Enter a string for the name of the transaction to run.
PARAM_PAIRS	Enter the name and value pairs (string name and value) for this transaction. Not used if the transaction has no parameters. Parameters are defined in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

This method returns a constant. Valid values are:

Value	*Description*
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

This PeopleCode example runs an asynchronous optimization transaction named SOLVE. It has no input or output parameters. The SOLVE transaction solves the exercise scheduling problem and puts the results into the QE_RWSM_EXERSCH table.

```
Local OptEngine &myopt;
Local integer &status;
&myopt = GetOptEngine("PATSMITH");
/* Run the SOLVE transaction asynchronously with input values.  */
&status = &myopt.RunAsynch("SOLVE");
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "SOLVE transaction failed.";
   Return;
End-If;
```

The following example shows the use of the DetailedStatus property.

```
Local integer &status;
&status = myopt.RunAsynch("SOLVE");
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```


RunSynch

Syntax

RunSynch*(TRANSACTION, PARM_PAIRS)*

Description

The RunSynch method requests the optimization engine to run the transaction in synchronous mode.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by RunSynch is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call RunSynch.

For example, RunSynch can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For RunSynch, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Parameters

Parameter	***Description***
TRANSACTION	Enter a string for the name of the transaction to run.
PARAM_PAIRS	Enter the name and value pairs (string name and value) for this transaction. Not used if the transaction has no parameters. Parameters are defined in the analytic type definition. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

This method returns a constant. Valid values are:

Value	***Description***
%OptEng_Success	Returned if method succeeds.

Value	Description
%OptEng_Fail	Returned if the method fails.

Example

The following PeopleCode example runs a synchronous optimization transaction named IS_MACHINE_AVAILABLE. It has these parameters:

- Input MACHINE_NAME to specify the machine.
- Inputs BEGIN_DATE and END_DATE to specify the time slot.
- Output AVAILABLE_FLAG to specify whether the machine is available in that time slot.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local datetime &begindate;
Local datetime &enddate;
&myopt = GetOptEngine("PATSMITH");
&machname = QEOPT_WRK.MACHINE_NAME.Value;
&begindate = QEOPT_WRK.BEGIN_DATE.Value;
&enddate = QEOPT_WRK.END_DATE.Value;
/* Run the IS_MACHINE_AVAILABLE transaction synchronously with input values.  */
&status = &myopt.RunSynch("IS_MACHINE_AVAILABLE",
 "MACHINE_NAME", &machname, "BEGIN_DATE", &begindate, "END_DATE", &enddate);
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "IS_MACHINE_AVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the IS_MACHINE_AVAILABLE transaction.  */
QEOPT_WRK.AVAILABLE_FLAG = &myopt.GetNumber("AVAILABLE_FLAG");
```

Or, the following example shows the use of the DetailedStatus property.

```
Local integer &status;
&status = myopt.RunSynch("SOLVE");
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

SetTraceLevel

Syntax

SetTraceLevel*(component,severity)*

Description

SetTraceLevel sets the severity level at which events are logged for a given component.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by SetTraceLevel is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call SetTraceLevel.

For example, SetTraceLevel can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For SetTraceLevel, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Parameters

Parameter	***Description***
component	Use one of the following PeopleCode constants: Opt_Engine, Opt_Utility, Opt_Datacache, or Opt_Plugin.
severity	Use one of the following PeopleCode constants. These options set the degree to which errors are logged. You can set the tracing levels differently for various parts of your program. This enables you to control the amount of trace information that your program generates. The following list shows the order of the severity, starting with the highest level. For example, %Severity_Error logs %Severity_Fatal, %Severity_Status, and %Severity_Error messages, while the system filters out other messages. Keep in mind that the higher the severity, the greater the performance overhead. • %Severity_Fatal • %Severity_Status • %Severity_Error • %Severity_Warn • %Severity_Info • %Severity_Trace1 • %Severity_Trace2

Returns

This method returns a constant. Valid values are:

Value	*Description*
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local datetime &begindate;
Local datetime &enddate;

&myopt = GetOptEngine("PATSMITH");

&status = &myopt.SetTraceLevel(%Opt_Engine, %Severity_Warn);
if &status = %OptEng_Fail then
   <example: notify user that set trace action has failed>
End-if;
```

ShutDown

Syntax

ShutDown()

Description

The ShutDown method requests the optimization engine to shut down.

If the optimization engine cannot be contacted for shutdown, the return status is %OptEng_Fail and the DetailedStatus property is OptEng_Not_Available.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by Shutdown is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call Shutdown.

For example, Shutdown can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For Shutdown, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Note. Before this method is called, CreateOptEngine or GetOptEngine must be called. Call ShutDown to shut down optimization engines even when running in Application Engine.

Parameters

None.

Returns

This method returns a constant. Valid values are:

Value	*Description*
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

This PeopleCode example shows an optimization engine being shut down:

```
Local OptEngine &myopt;
Local integer &status;
&myopt = GetOptEngine("PATSMITH");
/* Shut down the optimization engine */
&status = &myopt.ShutDown();
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "PATSMITH optimization engine shutdown failed.";
   Return;
Else
   QEOPT_WRK.MESSAGE_TEXT = "PATSMITH optimization engine shutdown successful.";
   Return;
End-If;
```

The following example shows the use of the DetailedStatus property:

```
Local integer &status;
&status = myopt.ShutDown();
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

OptEngine Class Properties

This section lists the optimization properties for the OptEngine PeopleCode class. The properties are listed in alphabetical order.

DetailMsgs

Description

The DetailMsgs property returns a list of messages generated by an optimization engine. Use DetailMsgs after you use the RunAsynch and RunSynch methods to check the status messages for an optimization transaction.

If the transaction fails, detailed messages are automatically shown to the user. If the transaction succeeds, warnings and informational messages may be generated by the transaction. Use this property to retrieve those messages and make them available to the user.

DetailMsgs provides a two-dimensional array containing the message set ID, the message number in the message catalog, and any arguments. Each row in the two-dimensional array has the following structure:

1. Message set ID.
2. Message number.
3. Number of message arguments.
4. Argument1.
5. Argument2.
6. Argument3.
7. Argument4.
8. Argument5.

A maximum of five arguments is supported for each message.

Note. To hold the property value returned, you need to declare an array of array of type *Any*.

Note. Before this method is called, you must call CreateOptEngine or GetOptEngine.

Example

```
Local OptEngine &myopt;
Local integer &status;
Local string &piid;

Local string &string;
Local array of array of any &arrArray;

&NEWLINE = Char(10);
&string = "";

&piid = GetRecord(Record.PSOPTPRBINST).GetField(Field.PROBINST).Value;
&myopt = GetOptEngine(&piid);

&status = &myopt.RunSynch("TEST_TRANSACTION");

If (&status = %OptEng_Success) then

&arrArray = &myopt.DetailMsgs;
For &iloop = 1 To &arrArray.Len

   &string = &string | &NEWLINE | MsgGetText(&arrArray [&iloop][1] /*message set*
/,
   &arrArray [&iloop][2] /*message id*/, "Message Not Found",&arrArray[&iloop][4],
   &arrArray [&iloop][5],&arrArray [&iloop][6],
&arrArray [&iloop][7],&arrArray[&iloop][8]);

End-For;

GetLevel0().GetRow(1).GetRecord(Record.QE_FUNCLIB_OPT).DESCRLONG.Value = &string;
End-If;
```

DetailedStatus

Description

The DetailedStatus property contains the detailed execution status of an OptEngine method after the method is executed.

Example

```
Local integer &status;
&status = myopt.ShutDown();
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```


OptBase Application Class

This PeopleCode application class is part of the PT_OPT_BASE application package. It establishes the basic framework for developing PeopleCode that invokes the Optimization PeopleCode plug-in. To use the plug-in, you develop a application class that extends the OptBase application class. OptBase contains the following types of methods:

- A set of base methods that you can extend for the purpose of handling input and output parameters.

  You can use them within any method you develop that corresponds by name to a transaction in an analytic type definition. These methods apply to the parameters that are defined for the transaction in the analytic type.

- A set of abstract placeholder methods that you can use to implement callback capability.

  You must extend these if you designate one or more records as callback records in your analytic type definition, even if you don't add any functionality to the methods.

- An abstract placeholder method, Init, that you can extend if you want to do any preprocessing before your first Optimization PeopleCode plug-in transaction runs.

Note. The analytic type definition to which these methods apply is the one that specifies this derived application class.

The CreateOptInterface function is the only optimization built-in function that you can use within an application class that you extend from the OptBase application class, or within PeopleCode that you call from that application class.

Optbase Callback Methods

PeopleSoft Optimization Framework has a built-in callback functionality when the OptInterface PeopleCode calls an Optimization PeopleCode plug-in transaction, it first determines whether you designated one or more records in your analytic type definition as callback records. For each callback record, the framework determines if any the record's database rows have been inserted, deleted, or updated since the optimization datacache was populated. If any changes have occurred, the framework propagates those changes to the datacache before invoking the transaction.

PeopleSoft provides methods that the framework uses to apply its callback functionality. In combination with the framework's callback changes, you might want to perform additional processing for your own purposes, including updating any derived data structures that are used by your optimization application. You can accomplish this by extending the callback methods and adding your own PeopleCode. Each callback method launches under different circumstances.

Note. Don't call any of these methods in your own PeopleCode. They're called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run within each method.

Following is a list of the abstract callback placeholder methods documented as part of the PT_OPT_BASE:OptBase application class:

- OptInsertCallback

  This method launches when the framework propagates to the datacache any database insertions encountered for a callback record.

- OptDeleteCallback

  This method launches when the framework propagates to the datacache any database deletions encountered for a callback record.

- OptPreUpdateCallback

  This method launches before the framework propagates each database update encountered for a callback record.

- OptPostUpdateCallback

  This method launches after the framework propagates each database update encountered for a callback record.

- OptRefreshCallback

  This method launches after the framework propagates all database deletions, insertions, and updates encountered for all callback records.

Important! If any record in your analytic type definition is designated a callback record, you must ensure that you extend all of the callback methods in your extended class, even if each extended method contains only a Return statement. Otherwise your Optimization PeopleCode plug-in will fail.

See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Records.

OptBase Class Methods

This section discusses the abstract base class placeholder methods for the PT_OPT_BASE:OptBase application class. The methods are listed in alphabetical order.

GetParmDate

Syntax

GetParmDate*(parmName,&parmVal)*

Description

The GetParmDate method retrieves a Date parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmDateArray

Syntax

`GetParmDateArray`*`(parmName,&parmVal)`*

Description

The GetParmDateArray method retrieves a Date array parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmDateTime

Syntax

`GetParmDateTime``(parmName,&parmVal)`

Description

The GetParmDateTime method retrieves a DateTime parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmDateTimeArray

Syntax

`GetParmDateTimeArray``(parmName,&parmVal)`

Description

The GetParmDateTimeArray method retrieves a DateTime array parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmNumber

Syntax

`GetParmNumber``(parmName,&parmVal)`

Description

The GetParmNumber method retrieves a Number parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmNumberArray

Syntax

```
GetParmNumberArray(parmName,&parmVal)
```

Description

The GetParmNumberArray method retrieves a Number array parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmInt

Syntax

```
GetParmInt(parmName,&parmVal)
```

Description

The GetParmInt method retrieves an Integer parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify an Integer variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmIntArray

Syntax

`GetParmIntArray``(parmName,&parmVal)`

Description

The GetParmIntArray method retrieves a Number array parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmString

Syntax

`GetParmString(parmName,&parmVal)`

Description

The GetParmString method retrieves a String parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmStringArray

Syntax

`GetParmStringArray(parmName,&parmVal)`

Description

The GetParmStringArray method retrieves a String array parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmTime

Syntax

GetParmTime*(parmName,&parmVal)*

Description

The GetParmTime method retrieves a Time parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmTimeArray

Syntax

GetParmTimeArray*(parmName,&parmVal)*

Description

The GetParmTimeArray method retrieves a Time array parameter value that passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

Init

Syntax

Init()

Description

The Init method launches when the CreateOptEngine built-in function loads an analytic instance that uses the Optimization PeopleCode plug-in.

Use this method to perform additional processing for your own purposes, including checking table data, or any functionality you want to apply before any plug-in transactions run. You accomplish this by adding your own PeopleCode to the extended method.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run before any other code in your extended class.

Note. If you don't extend this method, PeopleSoft Optimization Framework calls its base version from the OptBase application class.

Parameters

None.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptDeleteCallback

Syntax

`OptDeleteCallback`*`(&Record)`*

Description

The OptDeleteCallback method launches when PeopleSoft Optimization Framework propagates to the datacache any database deletions that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the deletion. You accomplish this by adding your own PeopleCode to the extended method.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	***Description***
&Record	Specifies a record variable that contains the keys of the data row to be deleted.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptInsertCallback

Syntax

`OptInsertCallback`*`(&Record)`*

Description

The OptInsertCallback method launches when PeopleSoft Optimization Framework propagates to the datacache any database insertion that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the insertion. You accomplish this by adding your own PeopleCode to the extended method.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	*Description*
&Record	Specifies a record variable that contains the new data row to be inserted.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptPostUpdateCallback

Syntax

`OptPostUpdateCallback`*`(&OldRecord,&NewRecord`*`)`

Description

The OptPostUpdateCallback method launches after PeopleSoft Optimization Framework propagates to the datacache any database update that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might have been affected by the update. You accomplish this by adding your own PeopleCode to the extended method. The parameters provide the previous and current content of the row.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	***Description***
&OldRecord	Specifies a record variable that contains the pre-update content of the data row that was updated.
&NewRecord	Specifies a record variable that contains the post-update content of the data row that was updated.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptPreUpdateCallback

Syntax

`OptPreUpdateCallback(&OldRecord,&NewRecord)`

Description

The OptPreUpdateCallback method launches before PeopleSoft Optimization Framework propagates to the datacache any database update that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the update. You accomplish this by adding your own PeopleCode to the extended method. The parameters provide the current and future content of the row.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	*Description*
&OldRecord	Specifies a record variable that contains the pre-update content of the data row to be updated.
&NewRecord	Specifies a record variable that contains the post-update content of the data row to be updated.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptRefreshCallback

Syntax

`OptRefreshCallback``()`

Description

The OptRefreshCallback method launches after PeopleSoft Optimization Framework propagates to the datacache all database insertions, deletions, and updates that it encounters for all callback records.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the modifications. You accomplish this by adding your own PeopleCode to the extended method.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

None.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDate

Syntax

SetOutputParmDate*(parmName,&parmVal)*

Description

Use the SetOutputParmDate method to pass a Date parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDateArray

Syntax

SetOutputParmDateArray*(parmName,&parmVal)*

Description

Use the SetOutputParmDateArray method to pass a Date array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDateTime

Syntax

`SetOutputParmDateTime`*`(parmName,&parmVal)`*

Description

Use the SetOutputParmDateTime method to pass a DateTime parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDateTimeArray

Syntax

SetOutputParmDateTimeArray(*parmName*,*&parmVal*)

Description

Use the SetOutputParmDateTimeArray method to pass a DateTime array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmNumber

Syntax

SetOutputParmNumber(*parmName*,*&parmVal*)

Description

Use the SetOutputParmNumber method to pass a Number parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmNumberArray

Syntax

SetOutputParmNumberArray(*parmName*,*&parmVal*)

Description

Use the SetOutputParmNumberArray method to pass a Number array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmInt

Syntax

`SetOutputParmInt``(`*`parmName,&parmVal`*`)`

Description

Use the SetOutputParmInt method to pass an Integer parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify an Integer variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmIntArray

Syntax

`SetOutputParmIntArray``(`*`parmName,&parmVal`*`)`

Description

Use the SetOutputParmIntArray method to pass a Number array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmString

Syntax

`SetOutputParmString`*`(parmName,&parmVal)`*

Description

Use the SetOutputParmString method to pass a String parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmStringArray

Syntax

SetOutputParmStringArray(*parmName*,*&parmVal*)

Description

Use the SetOutputParmStringArray method to pass a String array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	***Description***
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmTime

Syntax

SetOutputParmTime(*parmName*,*&parmVal*)

Description

Use the SetOutputParmTime method to pass a Time parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmTimeArray

Syntax

`SetOutputParmTimeArray`*`(parmName,&parmVal)`*

Description

Use the SetOutputParmTimeArray method to pass a Time array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	*Description*
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptInterface Class Methods

This section discusses the optimization methods for the OptInterface PeopleCode class. The methods are listed in alphabetical order.

Note. You can use the OptInterface class methods only within an application class that you extend from the OptBase application class, or within PeopleCode that you call from that application class. This ensures that the OptInterface PeopleCode runs only on the optimization engine.

ActivateModel

Syntax

ActivateModel*(ModelID,SolverSettingID)*

Description

The ActivateModel method designates the specified model and solver setting as active. The model and the solver are initialized and populated with data from the current analytic instance.

Note. This method fails if the specified model (and by extension, one of its solver settings) is already active. If you want to activate a different solver setting for the same model, you must first deactivate the model.

See Chapter 28, "Optimization PeopleCode," DeactivateModel, page 1567.

Parameters

Parameter	***Description***
ModelID	Specify the name of the optimization model you want to activate. This must be the name of one of the models associated with the analytic type definition.
SolverSettingID	Specify the name of the solver setting you want to activate. This is the name you specified for the solver setting in the analytic type definition.

Returns

This method returns a constant value. Valid values are:

Value	***Description***
%OptInter_Success	Returned if method succeeds.

Value	Description
%OptInter_Fail	Returned if the solver fails to solve the problem.

Example

```
Local integer &result;
Local OptInterface &oi = CreateOptInterface();

&result = &oi.ActivateModel("QE_PSA_MODEL", "abc");
```

ActivateObjective

Syntax

ActivateObjective*(Model_Name,Objective_Name)*

Description

Use the ActivateObjective method to activate the specified objective for an optimization model.

Parameters

Parameter	Description
Model_Name	Specify the name of the model.
Objective_Name	Specify the name of the objective.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the solver fails to solve the problem.

DeactivateModel

Syntax

DeactivateModel(*ModelID*)

Description

The DeactivateModel method detaches the solver from the specified model.

Parameters

Parameter	*Description*
ModelID	Specify the name of the optimization model you want to deactivate. This must be the name of one of the models associated with the analytic type definition.

Returns

This method returns a constant value. Valid values are:

Value	*Description*
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the solver fails to solve the problem.

Example

```
Local integer &result;
Local OptInterface &oi = CreateOptInterface();

&result = &oi.DeactivateModel("QE_PSA_MODEL");
```

DumpMsgToLog

Syntax

DumpMsgToLog(*LogSeverity*, *Message*)

Description

The DumpMsgToLog method writes the specified status message to the optimization engine log file, with the specified severity.

Parameters

Parameter	***Description***
LogSeverity	Specify the severity level of the message, as one of the following system constants: • %Severity_Fatal • %Severity_Status • %Severity_Error • %Severity_Warn • %Severity_Info • %Severity_Trace1 • %Severity_Trace2
Message	Specify as a string the text of the log message.

Returns

None.

FindRowNum

Syntax

FindRowNum(*&Record* [, *startrow* [, *endrow* [, *field_list*]]])

Where *field_list* is a list of field names in the form:

[*fieldname1* [, *fieldname2*]]...

Description

The FindRowNum method determines the row number of a row in the datacache rowset. You provide a record with key values, and this method finds the row with the same key values and returns its row number.

Parameters

Parameter	*Description*
&Record	Specify a record with the same structure as the records that comprise the rowset, with its key fields populated.
startrow	Specify as an integer the starting row number of the search. Specify 0 to search from the first row in the rowset.
endrow	Specify as an integer the ending row number of the search. Specify 0 to search through the last row in the rowset.
fieldname	Specify the name of a field in the input record which contains a value to be matched. You can specify one or more field names, in any order. **Note.** If you use this parameter, the fields specified here are used to search, instead of the record's key fields. Any value that doesn't correspond to a field name is ignored.

Returns

The row number of the row containing the specified key values, or 0 if no row is found.

Example

The following example searches the whole scroll to find the partial key OPT_SITE:

```
Local Record &rec = CreateRecord(Scroll.OPT_TRANSCOST);
Local Optineterface &oi;

&rec.OPT_SITE.value = "New York";
int nRowNum = &oi.FindRowNum(&rec, 0, 0, "OPT_SITE");
```

The following example searches from row 5 to row 15 with the full key values New York and San Jose:

```
Local Record &rec = CreateRecord(Scroll.OPT_TRANSCOST);
Local Optineterface &oi;

&rec.OPT_SITE.value = "New York";
&rec.OPT_STORE.value = "San Jose";
int nRowNum = &oi.FindRowNum(&rec, 5, 15);
```

GetSolution

Syntax

GetSolution*(ModelID,varArrayID,skipZero* [, *KeyFieldNames,KeyFieldValues* [, *&Solution*]])

Description

The GetSolution method retrieves the model solution values generated by the Solve method.

Parameters

Parameter	***Description***
ModelID	Specify as a string the name of the optimization model for which you want the solution. This is the name used for the model definition in Application Designer.
varArrayID	Specify as a string the name of the variable array being optimized. Your application documentation should provide this name.
skipZero	Indicate whether solutions with a value of zero should be fetched. This parameter takes a Boolean value: • True: Don't fetch solutions with a zero value. This can increase the performance of the GetSolution method if zero values aren't meaningful. • False: Do fetch solutions with a zero value.
KeyFieldNames and *KeyFieldValues*	Specify a set of key field names as an array of string and a set of key field values as an equal length array of ANY, with one key field value corresponding to each key field name. You use these arrays to restrict the set of returned solutions. Solutions are returned only for model variables with the specified key field values. **Note.** If you provide either of these arrays, you must provide both. You can include each parameter from the variable array at most only once.
&Solution	Specify a rowset to contain the solutions.

Returns

This method returns a constant value. Valid values are:

Value	***Description***
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the solver fails to solve the problem.

Example

```
Local array of string &strArray;
Local array of any &valArray;
Local integer &index;
Local Rowset &rowSet;
Local integer &result;
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local boolean &bSkipZero = True;

Local OptInterface &oi = CreateOptInterface();

&strArray = CreateArrayRept("", 0);
&valArray = CreateArrayAny();
&rowSet = CreateRowset(Record.QEOPT_VAL_X_WRK);

&strArray [1] = "EMPLID";
&valArray [1] = 1;
&strArray [2] = "ORDER_ID";
&valArray [2] = 23;

/* fetch only the part of the solution where  EMPLID = 1 and ORDER_ID = 23 */
&result = &oi.GetSolution(&modelId, &varArrayName,
 &bSkipZero, &strArray, &valArray, &rowSet);
```

GetSolutionDetail

Syntax

GetSolutionDetail*(ModelID,SolutionType,Name,&Solution)*

Description

The GetSolutionDetail method retrieves the model solution detail of the specified type generated by the Solve method. You can retrieve dual value, slack value, or reduced cost information.

Parameters

Parameter	***Description***
ModelID	Specify as a string the name of the optimization model for which you want the solution detail. This is the name used for the model definition in Application Designer.

Parameter	Description
SolutionType	Specify a system constant indicating the type of solution detail you want to retrieve. The value you specify here determines the content of the *Name* and *&Solution* parameters. • %OPT_DUAL: Retrieve the dual value attributes of the specified constraint block. • %OPT_SLACK: Retrieve the slack value attributes of the specified constraint block. • %OPT_RCOST: Retrieve the reduced cost attributes of the specified variable array.
Name	If you specified a *SolutionType* of %OPT_DUAL or %OPT_SLACK, specify here the name of a constraint block from the active model. If you specified a *SolutionType* of %OPT_RCOST, specify here the name of a variable array from the active model.
&Solution	Specify a rowset to contain the solution details. The rowset should have the same key fields as the constraint block or the variable array you specified with the *Name* parameter.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if solver fails to solve the problem.

Example

```
Local Rowset &dual_rowset;
Local integer &result;
Local OptInterface &oi = CreateOptInterface();
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local string &constrName = "Constraint_1";

/* fetch dual values for Contraint "Constraint_1"
 in a rowset based on the QEOPT_C1_WRK record */

&dual_rowset = CreateRowset(Record.QEOPT_C1_WRK);
&result = &oi.GetSolutionDetail(&modelId, %Opt_Dual, &constrName, &dual_rowset);
```


IsModelActive

Syntax

IsModelActive*(ModelID)*

Description

Use the IsModelActive method to determine if the model specified by *ModelId* is active before it is used.

Parameters

Parameter	*Description*
ModelID	Specify the model ID as a string. This is the name used for the model definition in Application Designer.

Returns

A Boolean value: true if the model is active, false otherwise.

RestoreBounds

Syntax

RestoreBounds*(modelID* [,*varArrayID*])

Description

The RestoreBounds method returns the bounding values of the specified variable array or arrays to the current settings in the specified model.

If you previously called the SetVariableBounds method with the *changeModelBounds* parameter set to true for any variable or variable array, those bounding values still apply.

Parameters

Parameter	*Description*
modelID	Specify as a string the name of the optimization model for which you want to restore the bounding values. This is the name used for the model definition in Application Designer.

Parameter	*Description*
varArrayID	Specify as a string the name of a variable array for which you want to restore the bounding values. Your application documentation should provide this name. If you don't specify a variable array name, the bounding values are restored for all variable arrays in the specified model.

Returns

%OptInter_Success if the method succeeds, %OptInter_Fail otherwise.

SetVariableBounds

Syntax

`SetVariableBounds``(`*`modelID, varArrayID, boundType, lowerBound, upperBound, &keyRecord`*
`[,` *`changeModelBounds`*`])`

Description

The SetVariableBounds method overrides the bounding values specified for a model variable array, or for a variable within the array.

Parameters

Parameter	*Description*
modelID	Specify as a string the name of the optimization model for which you want to override the bounding values. This is the name used for the model definition in Application Designer.
varArrayID	Specify as a string the name of the variable array being optimized. Your application documentation should provide this name.
boundType	Specify a system constant indicating which bounding values to override. The value you specify here determines how the *lowerBound* and *upperBound* parameters are applied to the specified model. • %OPT_LOWER_BOUND: Override only the lower bound as specified by the *lowerBound* parameter. The *upperBound* parameter is ignored. • %OPT_UPPER_BOUND: Override only the upper bound as specified by the *upperBound* parameter. The *lowerBound* parameter is ignored. • %OPT_BOUND_BOTH: Override both the lower bound and the upper bound as specified by the *lowerBound* and *upperBound* parameters, respectively.

Parameter	**Description**
lowerBound	Specify as a number the lower bound that should be applied to a variable or a variable array if the *boundType* parameter permits the override. You can also set this parameter to one of the following system constants:
upperBound	Specify as a number the upper bound that should be applied to a variable or a variable array if the *boundType* parameter permits the override. You can also set this parameter to one of the following system constants:
&keyRecord	Specify a record with the same key fields as the variable array being optimized. To override the bounding values specified for a single variable within the array, populate the record's key fields to specify the variable. To override the bounding values specified for the entire variable array, set all of the record's fields to a null value. **Note.** You must either provide values for all keys, or set them all to null values.
changeModelBounds	Specify a Boolean value: • true: Indicates that the specified model should be updated in memory to reflect the specified variable bounds. Any analytic instance that invokes this model from the active optimization engine is affected by these settings, which are propagated to the solver in memory. This is the default value if you omit this parameter. • false: Indicates that the specified model should not be updated in memory, and that the specified variable bounds apply only to the next time the Solve method is called.

Returns

%OptInter_Success if the method succeeds, %OptInter_Fail otherwise.

Example

```
Local Record &rec;
Local integer &result;
Local OptInterface &oi = CreateOptInterface();
Local float &objval = 0.0;
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local float &lb = 0.0;
Local float &ub = 0.0;

&rec = CreateRecord(Record.QEOPT_VAL_X_WRK);
&rec.QEOPT_RESINDEX.Value = 1;
&rec.QEOPT_SOLINDEX.Value = 2;
&rec.QEOPT_TIMEINDEX.Value = 3;

&result = &oi.SetVariableBounds(&modelId, &varArrayName,
 %Opt_Upper_Bound, &lb, &ub, &rec, False);
```


SetVariableType

Syntax

SetVariableType(*modelID*, *varArrayID*, *varType*)

Description

Use the SetVariableType method to change the data type of a model variable array.

Parameters

Parameter	*Description*
ModelID	Specify as a string the name of the optimization model for which you want to change the variable type. This must be the name of one of the models associated with the analytic type definition.
varArrayID	Specify as a string the name of the variable array for which you want to change the variable type. Your application documentation should provide this name.
varType	Specify one of the following system constants representing the new variable type: • %Opt_Var_Cont: Represents a continuous variable type, which can be any floating point value. • %Opt_Var_Bin: Represents a binary variable type, for which the value can be only 0 or 1. • %Opt_Var_Int: Represents an integer variable type, which can be any integer.

Returns

%OptInter_Success if the method succeeds, %OptInter_Fail otherwise.

Example

```
Local OptInterface &oi = CreateOptInterface();
Local string &varArrayName = "X";
Local integer &result;

&result = &oi.SetVariableType("QE_PSA_MODEL",  &varArrayName, %Opt_Var_Bin);

If (&result <> %OptInter_Success) Then
   &oi.DumpMsgToLog(%Severity_Status, "Failed to change variable type ");
End-If;
```

Solve

Syntax

`Solve(modelID,SolutionType [, &objValue [, name-value_pairs]])`

Where *name-value_pairs* is a list of solver setting parameter values in the form:

`[parmname1,parmvalue1 [, parmname2,parmvalue2]]...`

Description

The Solve method solves the specified model using the currently active solver settings, and provides an objective value as the solution output. You can override the solver setting parameters. The returned solution status is a predefined system constant.

Parameters

Parameter	***Description***
ModelID	Specify as a string the name of the optimization model you want to solve. This is the name used for the model definition in Application Designer.
SolutionType	Specify a system constant indicating the type of solution detail you want the model to be solved for. • %OPT_DUAL: Generate dual value attributes. • %OPT_SLACK: Generate slack value attributes. • %OPT_RCOST: Generate reduced cost attributes. You can also combine any or all of these system constants, by connecting them with a plus sign (+), for example: %OPT_DUAL + %OPT_RCOST.
&objValue	Specify a reference to a variable of type float. This variable contains the output objective value produced by the solver upon successfully solving the specified optimization model.
parmname and *parmvalue*	Specify a solver setting parameter ID and value to override the original value you specified for the solver setting in the analytic type definition. You can override any or all of the solver setting parameter values. See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Models for Optimization.

Returns

One of the following system constants:

%OptInter_Fail: The solver fails to solve the problem.

%Opt_Optimal: The solution is optimal.

%Opt_Infeasible: The solution is infeasible.

%Opt_Unbounded: The solution is unbounded.

%Opt_Timeup: The solver reached the time limit specified in the solver setting.

%Opt_Iterlimit: The solver reached the limit on the number of iterations specified in the solver setting.

%Opt_LP_Max_Sols: The solver generated maximum number of solutions without improvement.

%Opt_Idle: The solution shows no improvement in a specified time limit.

%Opt_Unknown: The solver status is unknown.

%Opt_MIP_NumSolutions: The specified number of solutions corresponding to an MIP solver reached.

%Opt_MIP_NumNodes: The specified number of nodes corresponding to an MIP solver reached.

%Opt_Aborted: The solver aborted.

%Opt_User_Exit: A user exit was encountered.

%Opt_First_LP_NoOpt: While solving an MIP, the first LP solution obtained was not optimal.

Example

Following is an example of the basic use of the Solve method:

```
Local OptInterface &oi = CreateOptInterface();

Local float &objval = 0.0;
Local integer &result;
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local integer &solType;

&solType = %Opt_RCost + %Opt_Dual + %Opt_Slack;

/* Solve the problem */
&result = &oi.Solve("QE_PSA_MODEL", &solType, &objval);

If & result = %Opt_Optimal Then
   &oi.DumpMsgToLog(%Severity_Warn, " Solution Status =  " Optimal !!!");
Else
   &oi.DumpMsgToLog(%Severity_Warn, " Solution Status =  " | &result );
End-If;
```

Following is an example of a solver setting parameter override:

```
Local OptInterface &oi = CreateOptInterface();
Local float &objval = 0.0;
Local integer &result;

/* This overrides the solver setting for MPS_Filename and generates
 an MPS file called myfile.mps instead of the name specified
 in the current solver setting parameter. */

&result = &oi.Solve("QE_PSA_MODEL", %Opt_Primal, &objval, "MPS_FileName",
 "myfile");
```

Chapter 29

Page Class

This chapter provides an overview of Page class and discusses the following topics:

- Shortcut considerations.
- Page object declaration.
- Scope of a Page object.
- Page class built-in function.
- Page class reference.

Understanding Page Class

Use the page class to manipulate a page in a component. The most common use of this class is to hide or display a page in a component, either based on field values or the level of security of the user.

Generally, the PeopleCode used to manipulate a page object is associated with PeopleCode in the Activate event.

Note. The page object should not be used until after the page processor has loaded the page: do not instantiate this object in RowInit PeopleCode, use it in PreBuild, PostBuild or Activate instead.

In addition, if you need to hide the current page, PeopleSoft recommends using the PreBuild event for your program.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Component Build Processing in Update Modes

Shortcut Considerations

An expression of the form

PAGE.*pagename.property*

is equivalent to `GetPage(`**`PAGE.`***`name`*`).`*`property`*. For example, the following two lines of code are equivalent:

```
PAGE.MANAGER_APPROV.DisplayOnly = False;
```

```
GetPage(PAGE.MANAGER_APPROV).DisplayOnly = False;
```

Data Type of a Page Object

Use the Page data type to declare Page objects. For example,

```
Local Page &MYPAGE;
```

Scope of a Page Object

A page object can be instantiated from PeopleCode only.

Use this object only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message subscription, a Component Interface, and so on.

In addition, the page object should not be used until after the page processor has loaded the page: do not instantiate this object in RowInit PeopleCode. Use it in PostBuild or Activate instead.

Page Class Built-in Function

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetPage

Page Class Properties

In this section, we discuss each Page class properties.

CopyURLLink

Description

Use this property to enable the http link for a component. Clicking this link copies the address (URL) of the page to the clipboard. This is a Pagebar property.

Note. If you select the Disable Pagebar checkbox for the component in Application Designer, you cannot set any of the Pagebar properties using PeopleCode. If you deselect the Copy URL Link checkbox for the component in Application Designer, you can still set this option using this property.

This property is read-write.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Page Definitions," Setting Use Properties

CustomizePageLink

Description

Use this property to enable Customize Page link for a component. Clicking this link enables the user to control the initial display of the component. This property takes a Boolean value, true, the user can control the display, false otherwise. This is a Pagebar property.

Note. If you select the Disable Pagebar checkbox for the component in the Application Designer, you cannot set any of the Pagebar properties using PeopleCode. If you deselect the Customize Page Link checkbox for the component in Application Designer, you can still set this option using this property.

This property is read-write.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Page Definitions," Setting Use Properties

DisplayOnly

Description

Use this property to make a page display-only, or to enable a page so the fields can be edited. This property takes a Boolean value: True if the page is display-only, False otherwise.

Note. You cannot enable or disable the current page.

The value of DisplayOnly for a page is restored to the value set in Application Designer when the user goes to another component. This means you must set DisplayOnly for the page each time a component is accessed.

This property is read-write.

Example

```
If &MYPAGE.DisplayOnly Then
   &MYPAGE.Visible = True;
Else
   &MYPAGE.Visible = False;
End-If;
```

HelpLink

Description

Use this property to enable the Help link for a component. Clicking this link accesses the online help PeopleBook entry for the current page. This is a Pagebar property.

Note. If you select the Disable Pagebar checkbox for the component in the Application Designer, you cannot set and of the Pagebar properties using PeopleCode. If you deselect the Help Link checkbox for the component in Application Designer, you can still set this option using this property.

This property is read-write.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Page Definitions," Setting Use Properties

Name

Description

This property returns a reference to the page name definition (a string value.)

This property is read-only.

Example

```
&MYPAGENAME = &MYPAGE.Name;
```


NewWindowLink

Description

Use this property to enable New Window link for a component. Clicking this link opens a new browser window with the search page for the current component. Users can view or enter data in the new window. This is a Pagebar property.

Note. If you select the Disable Pagebar checkbox for the component in the Application Designer, you cannot set any of the Pagebar properties using PeopleCode. If you deselect the New Window Link checkbox for the component in Application Designer, you can still set this option using this property.

This property is read-write.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Page Definitions," Setting Use Properties

Visible

Description

Use this property to set the visibility of a page. If this property is set to True, the page is visible. If this property is set to False, the page is not visible. When the visibility is changed, the component tabs and menus are also automatically changed to reflect the new settings. Menus and tabs return to their original values when the user navigates to another component. In addition, if a page is set to be hidden in the component definition, you can change the value of the page to be visible at runtime.

Oracle recommends using the PreBuild event if you need to hide the current page.

Important! Page visibility alone (as set by the Visible property) does not determine whether a user can see a page. Visibility plus the permissions to access the page together determine whether a specific page is displayed to a particular user.

Example

In the following example, a page is hidden based on the value of the current field.

```
If PAYROLE_TYPE = "Global" Then
   PAGE.JOB_EARNINGS.Visible = False;
End-If;
```


See Also

PeopleTools 8.51 PeopleBook: Security Administration, "Setting Up Permission Lists," Setting Page Permissions

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Component Definitions," Setting Page Attributes

Chapter 30

Portal Registry Classes

This chapter provides an overview of the portal registry and discusses how to:

- Use the PortalRegistry API.
- Use security.
- Work with ValidFrom and ValidTo.
- Do error handling.
- Understand the life cycle of a PortalRegistry object.
- View the PortalRegistry class hierarchy.
- Use content references.
- Understand naming conventions.
- Delete content considerations.
- Save content considerations.
- Declare a PortalRegistry object.
- Understand the scope of a PortalRegistry object.
- Use PortalRegistry classes (reference section).
- Understand PortalRegistry classes examples.

See Also

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Understanding Portal Technology"

Understanding the Portal Registry

The portal registry is a tree structure in which content for a portal is organized, classified, and registered. The PortalRegistry classes (API) are used to update the portal registry. The Portal Administration pages provide a GUI access to the PortalRegistry API. You can also access them using a PeopleCode program you write yourself. How you access the portal registry depends on the type of updates required. Your organization will likely use both methods of updating the portal registry. This chapter focuses on accessing the PortalRegistry classes using PeopleCode.

A portal is a website that helps you navigate to other web-based applications and content. The PeopleSoft Portal is a *business portal*. It is similar to general purpose portals, except that its main purpose is to help end-users be more effective in accessing information to perform their jobs.

Each PeopleSoft Portal is defined by one PeopleSoft portal registry. The PeopleSoft portal registry consists of a number of system tables and associated data in a PeopleSoft database. The portal registry must reside within one PeopleSoft database. There can be more than one portal registry in a PeopleSoft database, but only one portal registry is associated with a PeopleSoft Portal.

The portal registry consists of the following primary parts:

- Folders
- Content references
- Nodes

Folders and content references make up the majority of the portal registry, and provide a hierarchical tree structure to describe various content that is *registered* as part of a PeopleSoft portal.

Nodes provide a logical name to a specific webserver and database, so content can be registered independent of specific webservers. It is used when the portal servlet attempts to retrieve content—both internal PeopleSoft content as well as external content—and to assemble pages.

The primary function of the portal is to take a target URL (generally a URL for a PeopleSoft component) that comes in from a user's browser, and assemble a page with that content and any other content. The layout and what content to assemble on the page is defined by the *node template*, which is composed of HTML. The portal attempts to find the content reference associated (that is, registered) with the target URL to get the template, or uses a series of default templates if it cannot get the template from a content reference.

Folders

Folders are how a hierarchical structure is created within a portal registry. Each folder has a parent, except the root folder, which is the top-level folder in a portal registry. Each folder can also contain child folders and content references. Folders are roughly analogous to directories within a file system. A folder can be further defined by a number of attributes (description, security, when it expires, and so on) that are useful within the portal environment.

See Also

Chapter 30, "Portal Registry Classes," Adding a Folder, page 1811

Content References

A content reference is simply a reference to a URL. After you create an entry for a content reference in the portal registry, it's considered registered. A content reference can be further defined by a number of attributes (description, security, when it expires, and so on) that are useful within the portal environment.

There are a number of distinct types of content references that can be broken down into the following broad categories:

- Target

- Template
- Component

A *target* type of content reference describes a registered URL that a user might reference. Typically, these would be a PeopleSoft Pure Internet Architecture component, such as a page in a transaction. When a user references a URL, the URL is looked up in the registry to find the target content reference.

A *template* type of content reference describes what, if any, other content to put on the page, and where to place that content. The portal attempts to find a template for every URL that is requested.

A *component* type of content reference describes a component that is typically placed on a target page or homepage.

See Also

Chapter 30, "Portal Registry Classes," ContentReference Links, page 1721

Nodes

When a content reference is created to register some content (that is, a URL) the specific URI (that is, the scheme, webserver, and so on) can be specified in the content reference. However, this has the drawback that every time the URI for a content reference changes (the webserver name changes, and so on), the content reference must be changed. Nodes are a way to create a logical name for a specific webserver, scheme, and so on. When the content reference is created you can specify a node name.

For example, suppose the name of a webserver changed. If you don't use nodes, you must check all your content references and change them accordingly. If you use nodes, you have to change only the webserver name in one place, and all the content references that use that node now automatically reference the correct webserver.

Nodes can be optional for content where the specified URL is already a complete URL, such as for external content. Nodes can be categorized as:

- default local
- local
- non-local (remote)

See Also

Chapter 30, "Portal Registry Classes," Using Attributes, page 1819

Security

The same security mechanism is used for folders, content references, content reference links, tab definitions, pagelets, and user homepages. All of these items can be marked as public, owner accessible, or can have explicit PeopleSoft permission values set, including cascading the permissions to its child objects (that is, the child objects have the same permissions as the parent objects, when applicable.)

When marked as public, the item is accessible by any user. Public access cannot be cascaded, that is, passed down, to child objects.

When marked as owner accessible, an item is accessible only by the same user that created that item. Owner accessible cannot be cascaded, that is, passed down, to child objects.

Items can be marked as accessible by PeopleSoft permissions. This means that if a user is a member of a role, and the role contains the permission list that the item is also associated with, the user has access to that item. Additionally, when applied to folder permissions the permission can be cascaded. This means that any child of that folder, including a content reference, automatically has that permission added to its permission list.

Role-based security can applied to the portal objects (folders, content references and content reference links) using the RolePermissions collection.

Note. You can only use role-based security for content references, folders, and content reference links that are not components or iScripts.

You can specify which roles are allowed to access the objects. This works similarly to permission lists. If a user has a specified role, authorization is granted.

A role collection is the same as the PermissionValue collection, though there are additional properties on PermissionValue.

Attributes

Folders, content references, content reference links, PageletCategory objects, and Pagelets can have any number of attributes added to them. Attributes are simply name/value pairs. These name/value pairs are defined and used by many portal-aware applications, such as the search engine, navigational display, and so on.

See Chapter 30, "Portal Registry Classes," Using Attributes, page 1819.

Additional Portal Objects

Using the PortalRegistry, you can also add and customize other items in your portal, such as tabs, homepages, and favorites.

The TabDefinition is a homepage tab that has been defined to the portal. It is what an administrator creates when they want to define a new homepage tab. The TabDefinition is based on content references.

The User Homepage is a personalized homepage for a user. It contains all the tabs and pagelets the user has selected for their homepage. The User Homepage is based on folders.

A Favorite represents a content reference that the user wants to keep a shortcut to. It contains the name of the content reference, and the label the user wants displayed for this favorite.

A Pagelets is an area on a page that contains content, and the template used to specify the style for that content. Pagelets are a type of content reference.

Using the PortalRegistry API

The PortalRegistry API provides the entry point to a specific portal registry. Common administrative tasks include adding, deleting, and renaming the registry objects (that is, folders, content references, tabs, and so on.) Additionally, there are many properties associated with every registry object, and all of these properties can be accessed and modified.

You can use the PortalRegistry API in batch mode to make many changes to a portal at once. For example, suppose something changed in your security system. You could write your own PeopleCode program, using the PortalRegistry classes, in an Application Engine program and change the access for all your users at once.

You can also use the PortalRegistry API to exchange data with an external system. For example, suppose an external merchant had an area on your company's portal, and the information there must be updated. One way to do this is for the merchant to use an Application Message to send the data to your system, then a subscription program would update the portal using the PortalRegistry API.

Each PortalRegistry object represents one specific PortalRegistry in a system. An empty PortalRegistry object is initially gotten from the PeopleSoft Session object. You can then open an existing PortalRegistry to view or change existing content, create a new PortalRegistry, or delete an existing PortalRegistry.

Using Security

The PermissionValue object associated with a folder, content reference, and so on, as well as specific properties on the object, work together to form the security for an object. In this section, we discuss how to:

- Use object properties.
- Access objects.

Using Object Properties

The properties that set permissions for an object are:

- AuthorAccess
- PublicAccess

The AuthorAccess property determines whether the author of the object has access to the object.

The PublicAccess property determines whether the object is generally accessible or not. If this property is set to True, all users have access to the object.

These properties apply only to an object. They *cannot* be cascaded, that is, passed down to child objects.

The other object property used with security, the Authorized property, indicates whether a user is authorized to access an object. The value of this property depends on whether the user has access.

Accessing Objects

When you get a folder, content reference, content reference link, PageletCategory or Pagelet collection, only the items that the end-user is authorized to access are in the collection.

An object is contained in a collection is based on the following algorithm.

- If the object is marked as PublicAccess it is automatically accessible.
- If the object is marked as AuthorAccess and the current UserId is the Author it is accessible.
- If the current user is in a permission list (class) that is in the Permissions collection for that object.
- If the current user is in a permission list (class) that is in the CascadedPermissions collection for that object.
- If the current user has the Portal Administrator role.

When you access a content reference or folder using a valid name and one of the Find methods (FindCRefByURL, FindCRefByName, FindCRefForURL, or FindFolderByName) a content reference or folder is returned whether the user is authorized to it. When you use these methods, always check the Authorized property. This is the only property you can view from an object that an end-user isn't authorized to.

In addition, if you know the specific URL for a tab, you could specify that in the FindCRefByURL. The tab is returned whether the user is authorized to it, so you should always check the Authorized property.

Using PermissionValue, RolePermissionValue, Cascading Permissions and CascadingRolePermissions

To set non role-based permissions for a folder, content reference, content reference link, PageletCategory object, or Pagelet, you must access the PermissionValue collection for that object using the Permissions property. The PermissionValue objects refer to permission list values that already exist on the system, such as ALLPANLS, CUSTOMER, EMPLOYEE, and so on.

To set role-based permissions for a folder, content reference, or a content reference link that are not component or iScript, you must access the RolePermissionValue collection for that object using the RolePermissions property. The PermissionValue objects refer to role-based permission values that already exist on the system.

Note. The PORTAL_PAGELETS folder is the parent folder for all PageletCategories. Its security is set to public. PeopleSoft does not recommend cascading any permissions from this folder object. Cascade permissions only from the pagelet category (folder).

Both the PermissionValue collection and the RolePermissionValue collection return PermissionValue objects. Use the PermType property to determin if a particular PermissionValue is role-based or not.

For every PermissionValue object, you can chose whether all the child folders and content references of this folder have the same permissions. This is called *cascading.* You cascade permissions by setting the Cascade property to True.

There are two types of PermissionList collections you can access. One is returned by the Permissions and RolePermissions properties, the other by the CascadedPermissions and CascadedRolePermissions properties.

Permissions and RolePermissions

The Permissions and RolePermissions properties return a collection containing the permissions set at the current level, that is, set with that folder or content reference. You can add PermissionValues to this list. Use the Cascade property to cascade the permission you add to child folders and content references.

Note. Folders can contain other folders, but the other objects that use the Permissions property can't contain themselves, that is, content references can't contain other content references, and so on. Therefore, the Cascade property is applicable only to folders, not to any other object.

CascadedPermissions and CascadedRolePermissions

The CascadedPermissions and CascadedRolePermissions properties return the a collection for *all* the permissions for an object. It contains any permission list values set in any parent folder.

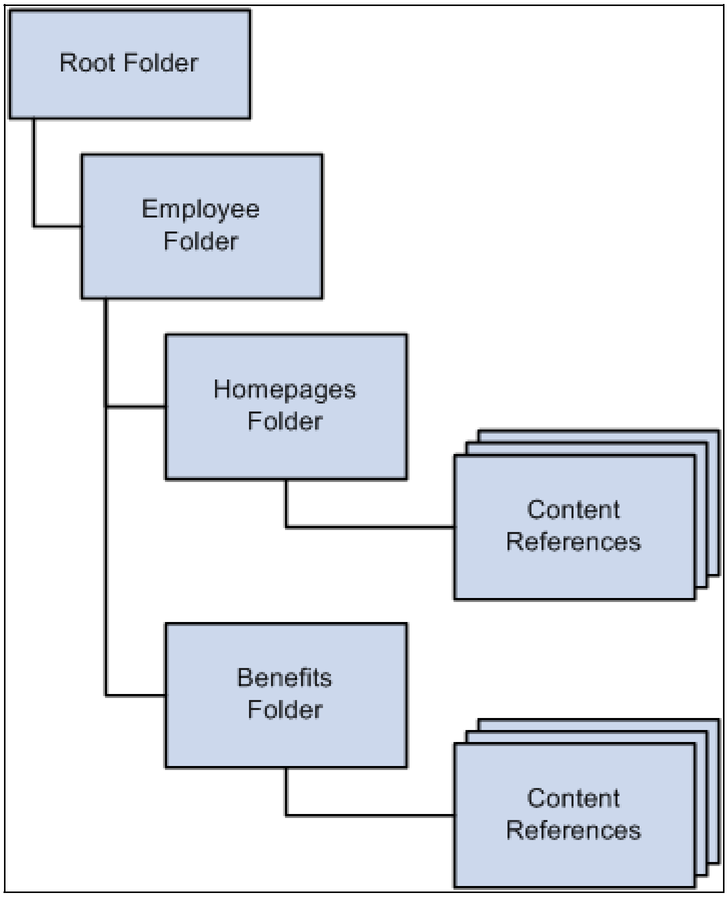
Note. You *cannot* add any PermissionValues to a collection returned by the CascadedPermissions or CascadedRolePermissions property. You can add values only to the collection returned by the Permissions or RolePermissions property.

You can add PermissionValues for a child folder, but you *cannot* delete any of the existing ones that are cascaded. Permissions are augmented, not overwritten, that is, the permissions are the sum of both the parent permissions and whatever is set at the current level.

Therefore, keep the following in mind when working with PermissionValues:

- Set permissions only at the level where they're needed.
- Cascade permissions only when necessary.
- Generally, don't set cascaded PermissionValues for the root folder.
- Only the values in a PermissionValue object cascade. Properties on the folder itself (such as PublicAccess or AuthorAccess) do *not* cascade.

For example, suppose your PortalRegistry had the following hierarchy:



PortalRegistry hierarchy example

The following permission list values are assigned to the PermissionValue objects for the Employees folder, and both of them are cascaded:

- EMPLOYEE
- MANAGER

The Homepages and Benefits folders have the exact same permissions as the Employees folder, that is, all users associated with these two permission lists have access to these folders.

Suppose you decide that you don't want the permission list EMPLOYEE to access the Benefits folder. If you delete the EMPLOYEE PermissionValue from the Benefits PermissionValue collection, you have *not* altered the permissions. The permissions set at the parent folder, that is, at the Employees folder, can't be deleted, they can only be added to.

To make this change, you must:

- Change the Cascaded property for the EMPLOYEES PermissionValue in the Employees folder to False
- Add EMPLOYEES as a PermissionValue object to the Homepages folder.

See Also

Chapter 30, "Portal Registry Classes," Cascade, page 1711

Chapter 30, "Portal Registry Classes," Setting Permissions Using the PermissionValue Object, page 1817

PeopleTools 8.51 PeopleBook: Security Administration, "Setting Up Permission Lists"

Working With ValidFrom and ValidTo

Folders, content references, content reference links, and tab definitions have both ValidFrom and ValidTo dates. What's the difference and how are they used in the PortalRegistry API?

- A ValidFrom date is when something begins.
- A ValidTo date is when something ends.

For example, suppose your HR department has a page describing the benefits for your employees, and that page changed every calendar year. This means the page for the year 2000 has a ValidFrom date of 01/01/2000 and a ValidTo date of 12/31/2000. The benefits page for the year 2001 has a ValidFrom date of 01/01/2001 and a ValidTo date of 12/31/2001.

Folders, content references, content reference links and tab definitions are returned in their collections regardless of the ValidTo and ValidFrom dates. You must take these dates into account and only display to the end-user those values that should be seen.

In addition, a ValidFrom should always come *before* a ValidTo date. If they are set incorrectly, you receive a runtime error.

For all newly created folders, content references, content reference links and tab definitions, the default value for both these properties is Null (""), that is, they begin immediately and do not expire.

In addition, you can also check the IsVisible property to see if a portal object is viewable. IsVisible verifies if an object is Hidden, as well as if the ValidTo and ValidFrom dates are within the specified dates.

Doing Error Handling

All errors for the PortalRegistry classes, like the other APIs, are logged in the PSMessages collection, instantiated from a session object.

The PortalRegistry classes log errors that occur with methods immediately, and errors that occur with properties only after a method is executed.

For example, suppose you specified an invalid name when you were trying to delete a folder. The method (DeleteItem) returns False, and the error is logged in the PSMessages collection immediately.

Now suppose you created a new folder, and specified an invalid ValidTo date. The error won't be logged in the PSMessages collection until you tried to save your changes.

When you want to check for errors depends on your application. When users are entering data dynamically, and your program is registering their data in the portal, you may want to check for errors often. If you're using a batch program, you may want to check for errors only after the Open, Save, Insert, and Delete methods.

Most methods return a Boolean value indicating success or failure. After the failure of a method, you may want to check the PSMessages collection to determine the exact error.

```
Local ApiObject &MySession;
Local ApiObject &ErrorCol;
Local ApiObject &FolderCol, &Folder, &Registry;
Local Boolean &Open;/* Access the current session */

&MySession = %Session;
If &MySession Then
   /* connection is good */

&Registry = &MySession.GetRegistry();

&Open = &Registry.Open("CUSTOMER");

If &Open Then
   /* Registry opened successfully */
   /* do processing */
Else
   /* Do error checking */

&ErrorCol = &MySession.PSMessages;
For &I = 1 to &ErrorCol.Count
   /* do processing */
End-For;

   End-If;
Else
   /* do processing for no connection */
End-If;
```

See Also

Chapter 38, "Session Class," Error Handling, page 2180

Understanding the Life Cycle of a PortalRegistry Object

At runtime, there are certain things you want to do with a PortalRegistry object, like getting an instance of it, adding tabs, editing content references, and so on. The following is an overview of this process. These steps are expanded in other sections.

1. Perform one of these steps.

   a. Execute the GetPortalRegistry method on the PeopleSoft session object to get a PortalRegistry object, or

   b. Use the FindPortalRegistries method on a session object to get a collection of all PortalRegistries. Select a PortalRegistry from the collection.

2. Open the PortalRegistry, using the portal name.

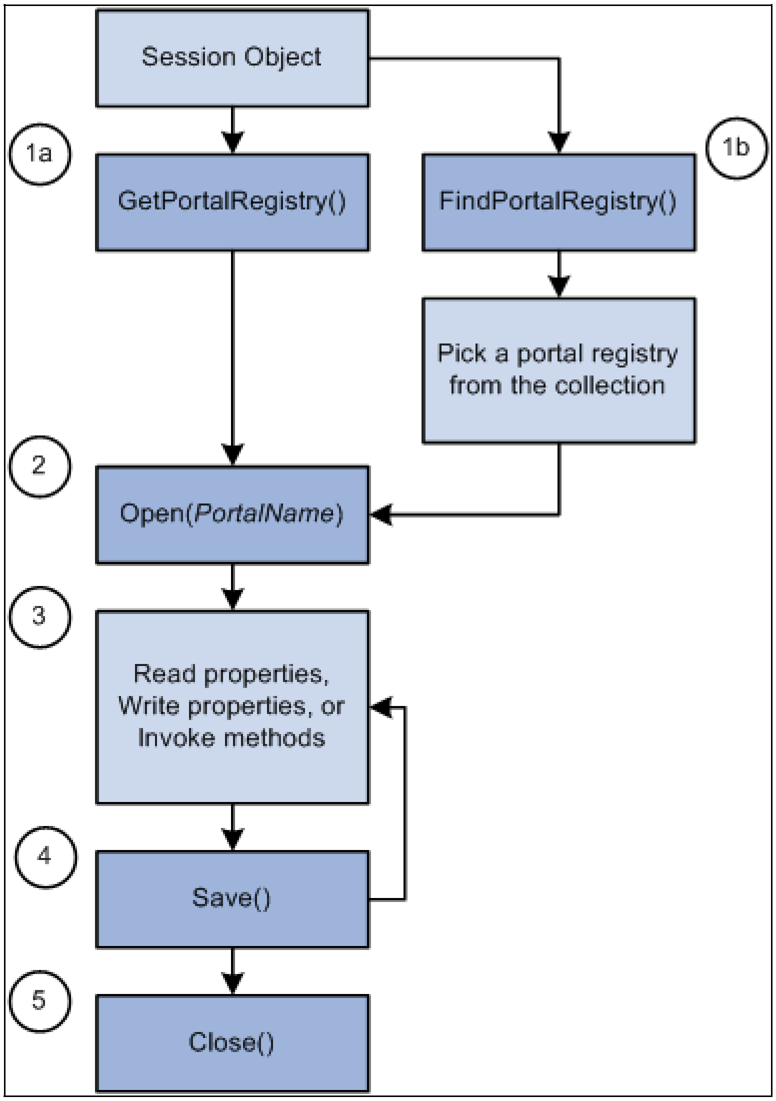
3. After you have an open PortalRegistry object, you can do various actions such as:
   - Add content references to the existing folders.
   - Add folders and the content references for them.
   - Add or change Nodes.
   - Modify the PortalRegistry properties.
   - Modify the existing content references, folders, tabs, homepages, and so on.
4. After you make your changes, you must save your work. The Save method must be executed at the appropriate level, such as at the folder level for changes to a folder, at the PortalRegistry level for changes to the default template, and so on.

   A property value change isn't committed to the database until the parent object is saved.

   Some methods commit data to the database only when the parent object is saved. However, other methods cause data to be committed to the database as soon as they are executed, like DeleteItem on a folder. Methods that automatically make database changes are noted as such in their description.

   See Chapter 30, "Portal Registry Classes," Saving Content Considerations, page 1609.
5. When you finish all operations for a PortalRegistry, close the object.

Note. PeopleSoft recommends that you open and close every PortalRegistry in a single PeopleCode event. You shouldn't open a PortalRegistry object and keep it open across multiple events. You should also keep only one PortalRegistry object open at a time.



Life cycle of a PortalRegistry object

Examples of using the PortalRegistry objects are provided at the end of this chapter.

See Also

Chapter 30, "Portal Registry Classes," PortalRegistry Classes Example, page 1808

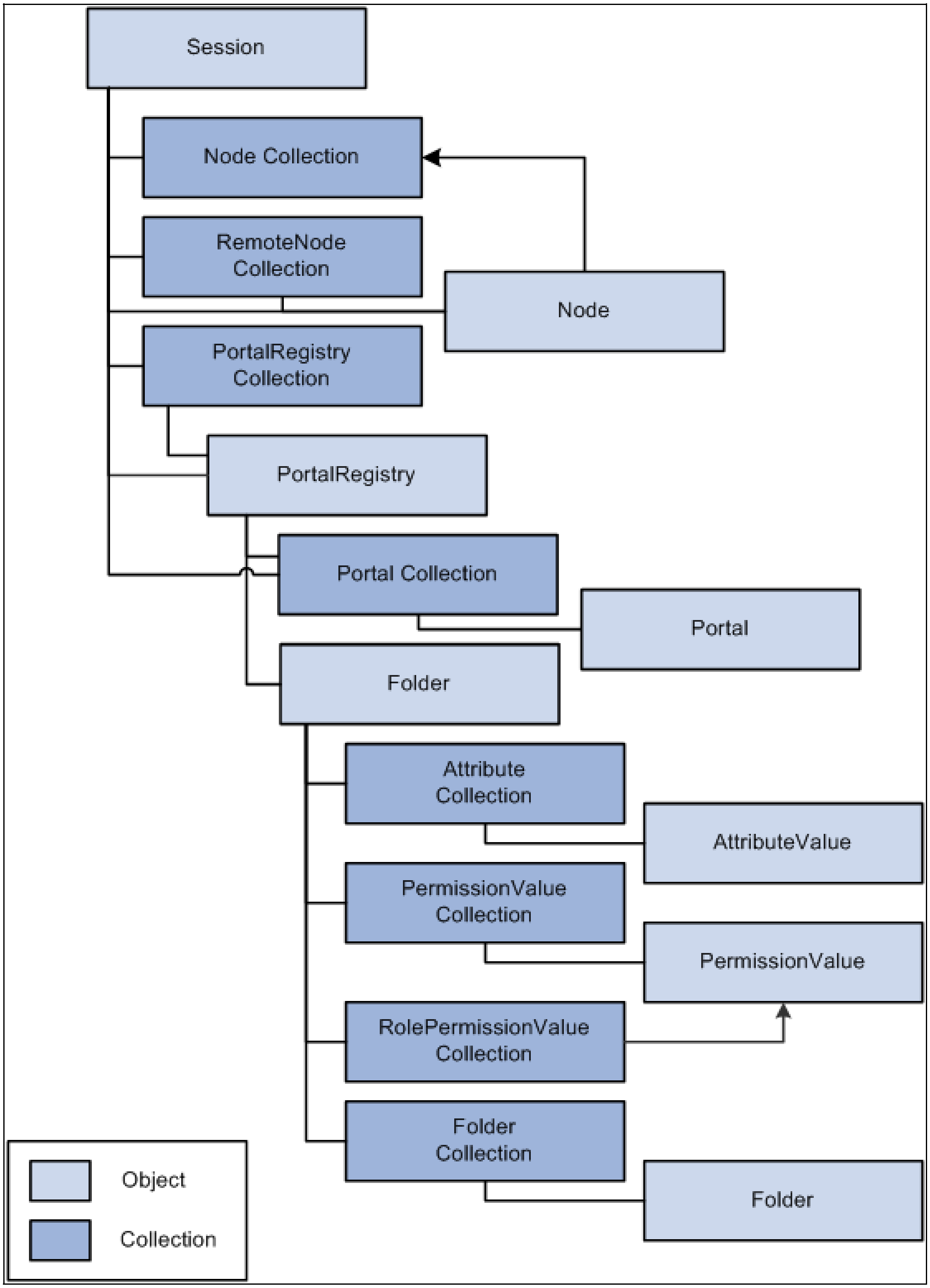
Viewing the PortalRegistry Class Hierarchy

There are many different classes used with the PortalRegistry API. The following flowchart shows the different classes, and how they are interrelated.

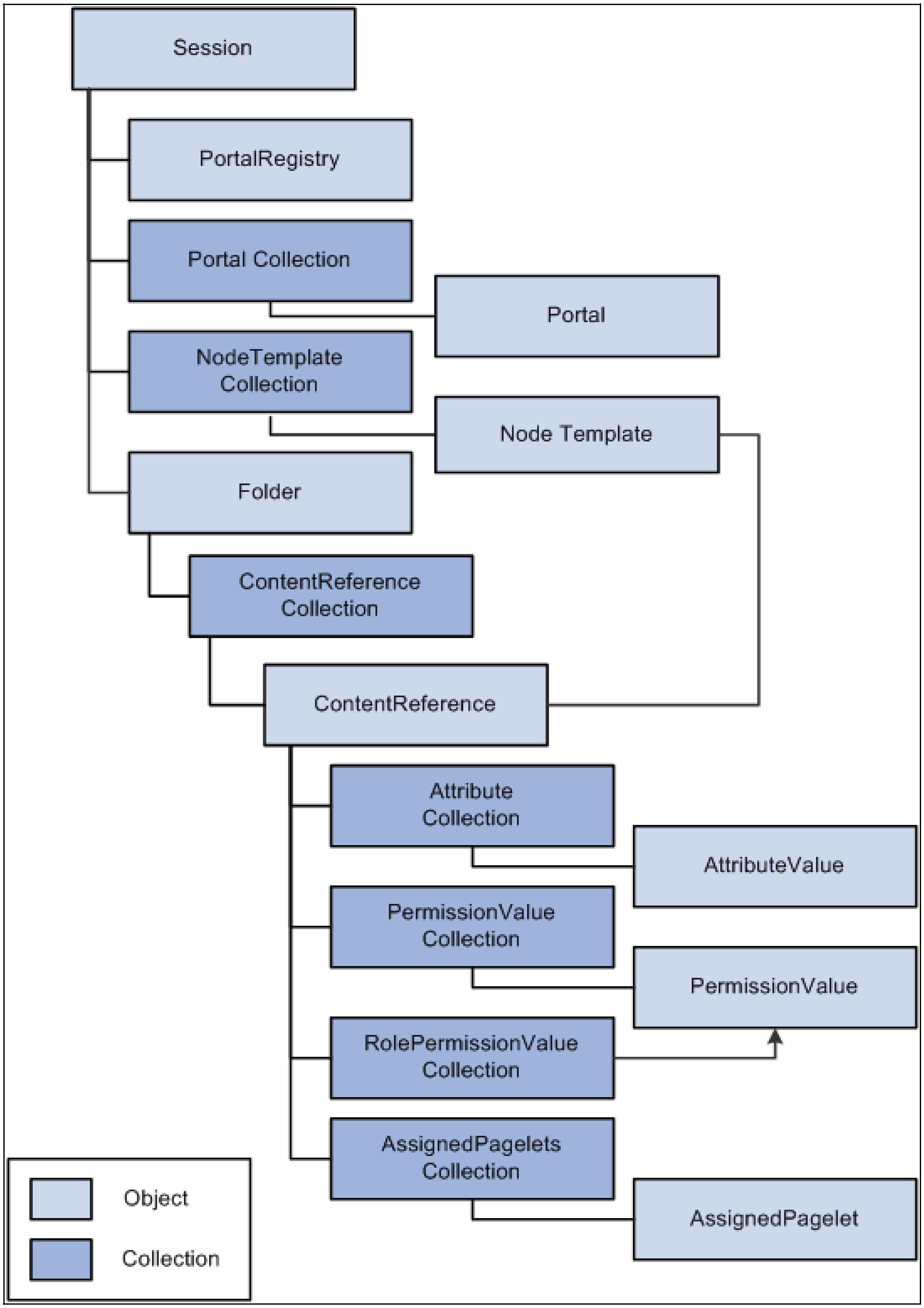
Hierarchy Considerations

The following are considerations for the representation of the hierarchy:

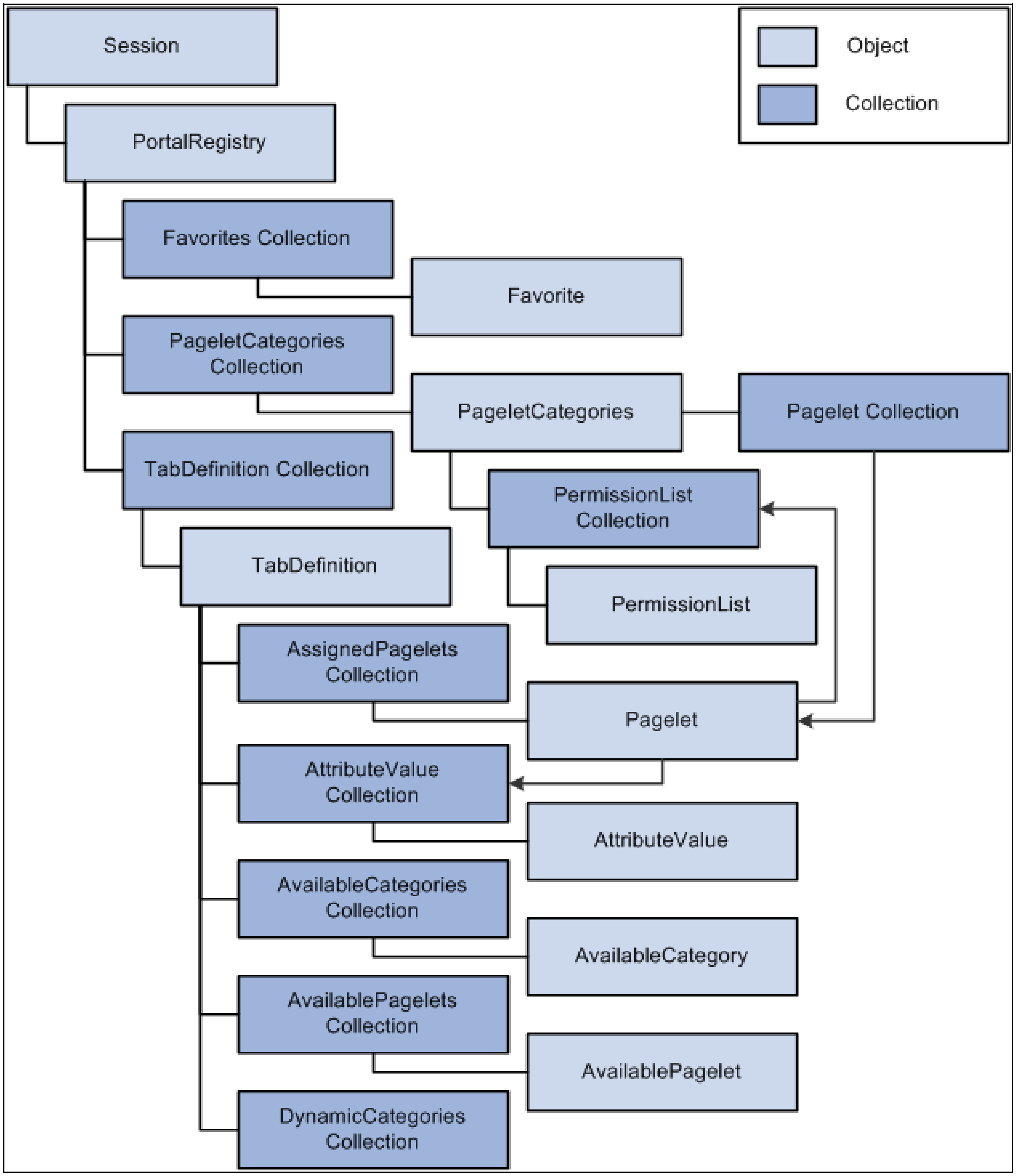
- The flowchart shows only two levels of folders. In reality, you can embed as many levels of folders as you need.
- From a PortalRegistry object, you must access the root folder before you can access the sub-folders for that PortalRegistry.



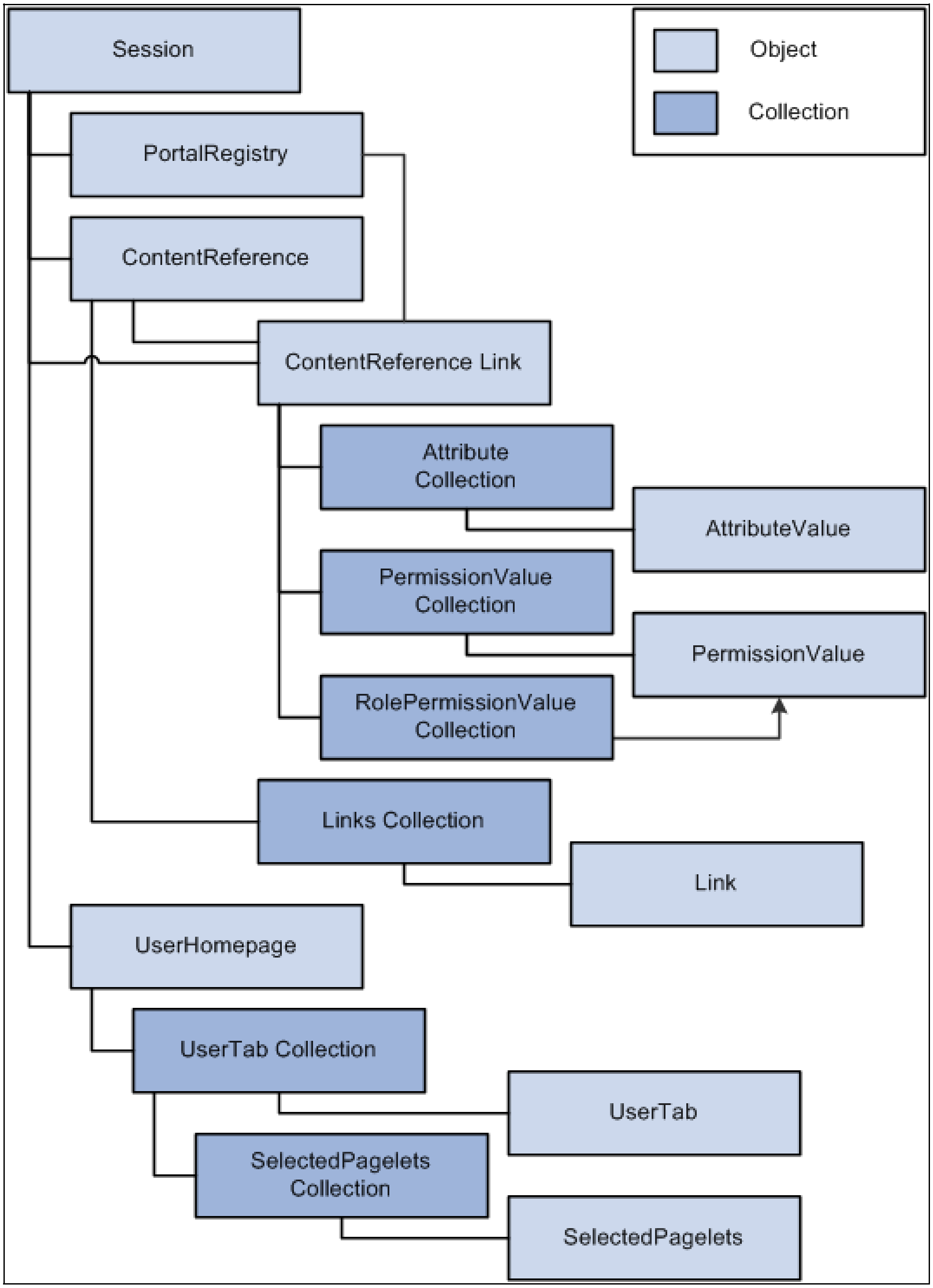
PortalRegistry class hierarchy (part 1 of 4)



PortalRegistry class hierarchy (part 2 of 4)



PortalRegistry class hierarchy (part 3 of 4)



PortalRegistry class hierarchy (part 4 of 4)

Using Content References

Content references have a number of properties, but several properties work together to define the type of Content reference. The values of these properties are interdependent, that is, the value of one indicates the values of others. The properties are:

- UsageType
- TemplateType
- URLType
- Nodes and URL

UsageType

This is the primary specifier for the type of content reference. There are a number of different types of content references, but all content references can be categorized into the following major types:

- Target
- Template
- Portal Component

Target

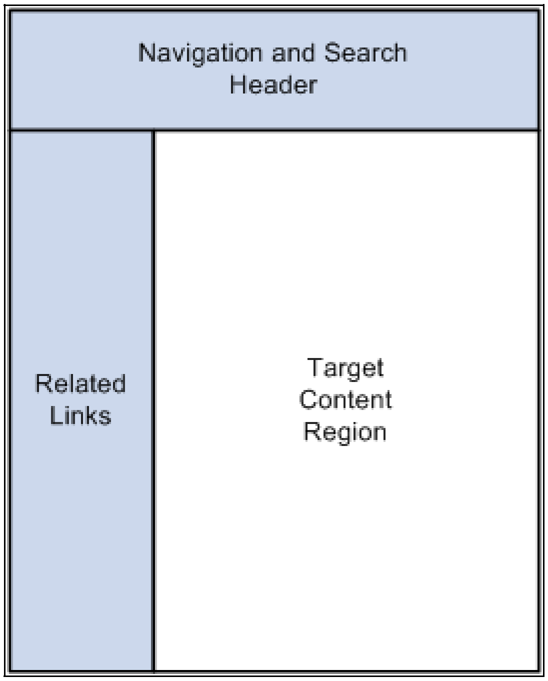
The *target* is the page referenced by a URL in the user's browser. This is the main transaction or page that the user wants, and the portal may place other content around this target page based on the template. The template describes any other content, and where to place it, on the page. The template is either gotten from the content reference or a set of default templates.

Content reference links can only be created for content references that have a type of target.

Template

A portal *template* defines how the portal creates a user's page. It's an HTML document that describes the content and where the content is placed. The template specifies one target and zero or more portal components.

The portal template in the following example is comprised of four separate template components: one for the navigation and search header, one for related links, one for the target content region, and one for the overall template which specifies where the other components should be placed. At runtime, the target content region would be filled by the HTML returned by the target page, as would the other template component regions.



A portal template

See *PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies*, "Working with Portal Templates."

Portal Component

A *portal component* is an HTML document, or something that produces an HTML document. The portal component must be defined within a template.

A portal component could be one of a Homepage tab, component reference, or Homepage Pagelet.

UsageType Values

The following table matches the general types of target, template, and portal component to the actual values of the UsageType property.

General Type	UsageType Value
Target	Target (TARG)
Template	Frame template (FRMT) or HTML template (HTMT)
Pagelet	Pagelet (HPGC)
Homepage Tab	Homepage Tab (HPGT)

More specifically:

- A UsageType value of TARG specifies a content reference that is a target.
- A UsageType value of FRMT specifies a content reference that is an HTML frame template.
- A UsageType value of HTMP specifies a content reference that is an HTML template.
- A UsageType value of HPGC specifies a content reference that is a PeopleSoft homepage component (pagelet).
- A UsageType value of HPGT specifies a Homepage tab.
- A UsageType value of LINK specifies a content reference link.

TemplateType

This property is valid only when the UsageType property is a target. For target type content references (TARG) this controls whether the portal looks for and uses a template to wrap the target.

TemplateType	*Meaning*
NONE	There is no template for this target
HTML	There is some kind of HTML template for this target

URLType

This property gives information about what format the URL is in.

URLType Value	*Meaning*
UEXT	URL points to a non-PeopleSoft (external) URL
UMPG	URL points to a PeopleSoft mobile page
UPGE	URL points to a component
UPHP	URL points to a homepage tab
UPTM	URL points to a template
USCR	URL points to an iScript
UGEN	URL points to a generic PeopleSoft URL.

The URL property is always required (it's one of the parameters for the InsertItem method.) The format of this parameter (or property) depends on the other properties.

See Also

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Understanding Portal Technology"

Nodes and URL

Nodes and the URL property work together and are interrelated. The node is how to 'register' a logical name for a webserver (the webserver name, on the servlet, and so on) not the actual details. This way, content references don't change when a webserver changes.

For example, suppose you had content that you referenced on the HRMS webserver. However, the machine name for that server changed. You can change the URI of the Node, instead of changing every content reference that referred to that content.

At least one node must be specified as the default local node. Nodes are also required for PeopleSoft components and iScripts.

Considerations When Using Nodes and the URL Property

When a content reference is created it is 'registered' with its URL (the portal, and others, typically find a content reference by its URL). The node is used to create a logical name for the webserver, servlet, and so on, so these details are not included in a content reference's URL. When details of a webserver change, such as at installation time, only the URI for the content provider must change. You don't have to change the URL for any content references.

When the node is specified, the content provider's URI is concatenated with the URL property.

The format of the URL property depends on the content it's pointing to.

Summary

The following table summarizes the interrelations between the different content reference properties.

UsageType	*TemplateType*	*StorageType*	*URLType*
Target (TARG)	HTML	Remote (RMTE)	Component (UPGE)
Target (TARG)	HTML	Remote (RMTE)	Internet Script (USCR)
Target (TARG)	HTML	Remote (RMTE)	External (UEXT)
Target (TARG)	NONE	Remote (RMTE)	Component (UPGE)

UsageType	***TemplateType***	***StorageType***	***URLType***
Target (TARG)	NONE	Remote (RMTE)	Internet Script (USCR)
Target (TARG)	NONE	Remote (RMTE)	External (UEXT)
Frame template (FRMT), HTMP template	NONE	Remote (RMTE)	Internet Script (USCR)
Frame template (FRMT), HTMP template	NONE	Local (LOCL)	N/A
Homepage (HPGT)	NONE	Local (LOCL)	N/A
Template component (TMPC), Homepage component (HPGC)	N/A	Remote (RMTE)	Component (UPGE)
Template component (TMPC), Homepage component (HPGC)	N/A	Remote (RMTE)	Internet Script (USCR)
Template component (TMPC), Homepage component (HPGC)	N/A	Remote (RMTE)	External (UEXT)
Template component (TMPC), Homepage component (HPGC)	N/A	Local (LOCL)	N/A

Naming Conventions

If you create two content references or with the exact same URL, the second one fails.

For example, suppose you create an external content reference with a URL of www.peoplesoft.com. Then you create a second content reference that has a node of PeopleSoft, whose URI is www.peoplesoft.com. The creation of the second content reference fails because the URL already exists.

If you have multiple nodes with the same URI, FindCRefByURL looks for the specified content reference with all those nodes.

If you have multiple nodes with the same URI, but no registered content references, the system uses the template from the alphabetically first node it finds.

Pagelet, node, and portal registry names can consist of any combination of letters, digits and underscores, but they must not contain any spaces or begin with a digit.

Content reference link names cannot have start with numbers, and can not have special characters and spaces.

Deleting Content Considerations

Be extremely careful when you delete any content. There may be more than one object relying on the content you delete.

- If you try to delete a template currently used by a content reference, the node template is set as the default template for the portal, you receive an error when you try to save the item.
- If you delete a homepage template for a user, the system tries to use the default user's template first, before resorting to the portal default template.
- If you try to delete a content provider that is currently used by a content reference, you receive an error and cannot save the PortalRegistry object.

Warning! If you delete a folder, you delete *all* content in the folder. If you delete a folder that contains other folders, that is, a parent folder, all the child folders, and all the content references are deleted.If you delete a PortalRegistry, you delete *everything.* Your entire PortalRegistry is gone, all the folders, content references, templates, and so on. Do not delete a PortalRegistry object unless you are absolutely certain you want to.

Saving Content Considerations

The following PortalRegistry classes have Save methods:

- PortalRegistry
- Folder
- Content reference
- Content reference links
- TabDefinition
- PageletCategory
- Pagelet
- Homepage
- Favorites collection

This means if you change a folder, you must save the folder. If you change a folder and only save at the PortalRegistry level, your changes are *not* saved.

Some classes do *not* have a Save method. For these classes, you must save the parent object.

- If you change a node, node template, or remote portal, you must use the PortalRegistry Save method.
- If you change an AttributeValue you must use the Save method with the item that contains the AttributeValue (folder, content reference, PageletCategory, or Pagelet).
- If you change a PermissionValue, you must use the Save method with the item that contains the PermissionValue (folder, content reference, PageletCategory or Pagelet).
- If you change a UserTab or a SelectedPagelet, you must use the Homepage Save method.

Data Type of a PortalRegistry Object

PortalRegistry objects, and all objects instantiated from a PortalRegistry object, are declared as an ApiObject data type. For example:

```
Local ApiObject &MyRegistry;

Local ApiObject &MyFolder, &MyAttributeValue;
```

Note. PortalRegistry objects can only be declared as Local.

Considerations Using Local Variables

When a local variable has a reference to an object and the end-user clicks the Back button on a browser, the local variable is set to NULL. This is always logged in the trace file.

Considerations Using Global Variables

Global variables are not available to a portal or applications on separate databases. They are available only on applications and Portals in the *same* database.

Scope of a PortalRegistry Object

A PortalRegistry object can be instantiated from the following language environments:

- PeopleCode
- C/C++
- Java

See Also

Chapter 30, "Portal Registry Classes," PortalRegistry Classes Example, page 1808

Portal Registry Reference

The following sections provide more detail of the properties, methods, and other objects that you can use with a PortalRegistry object. If you don't want to programmatically change a portal registry, you can use the PeopleTools, Portal pages instead.

See Also

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Understanding Portal Technology"

Session Class Methods

PortalRegistry objects don't have any built-in functions. They are instantiated from a session object. In this section, we discuss the Session Object methods. The methods are listed in alphabetical order.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetSession

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "System Variables," %Session

FindPortalRegistries

Syntax

FindPortalRegistries*(Name)*

Description

The FindPortalRegistries method returns a reference to a PortalRegistry Collection filled with zero or more PortalRegistry objects matching the *Name* parameter. The *Name* parameter takes a string value.

You can use a partial key to get a smaller subset of the PortalRegistry collection. For example, to get a collection of all the PortalRegistry objects whose names start with the letter "B", specify just the letter B for *Name*:

```
&MyColl = &MySession.FindPortalRegistries("B");
```

Parameters

Parameter	Description
Name	Specify the name of the PortalRegistry object to find. This parameter takes a string value.

Returns

A PortalRegistry Collection object.

Example

The following example returns a collection with references to all PortalRegistry objects.

```
Local ApiObject &MySession;
Local ApiObject &MyColl;

&MySession = %Session;
&MyColl = &MySession.FindPortalRegistries("");
```

GetActualRemoteNodes

Syntax

GetActualRemoteNodes()

Description

Use the GetActualRemoteNodes method to return a collection of remote nodes for the session.

Note. This method will only return remote nodes.

Parameters

None.

Returns

A reference to a collection of remote nodes if successful, null otherwise.

Example

```
&remoteNodeColl = %Session.GetActualRemoteNodes();
```


See Also

Chapter 30, "Portal Registry Classes," RemoteNode Collection, page 1652

GetLocalNode

Syntax

GetLocalNode()

Description

Use the GetLocalNode method to return a reference to the node defined as the local node for this session.

Parameters

None.

Returns

A reference to a node object if successful, Null otherwise.

Example

```
&Node = %Session.GetLocalNode();
```

See Also

Chapter 30, "Portal Registry Classes," Node Class, page 1645

GetNodes

Syntax

GetNodes()

Description

Use the GetNodes method to return a collection of both local and remote nodes for the session.

Parameters

None.

Returns

A reference to a collection of nodes if successful, Null otherwise.

Example

```
&NodeColl = %Session.GetNodes();
```

See Also

Chapter 30, "Portal Registry Classes," Node Collection, page 1649

GetPortalRegistry

Syntax

GetPortalRegistry()

Description

The GetPortalRegistry method returns an empty PortalRegistry object. You can then open or delete an existing PortalRegistry, or create a new one.

Parameters

None.

Returns

An empty PortalRegistry object.

Example

```
Local ApiObject &MyPortal;

&MyPortal = %Session.GetPortalRegistry();
&PORTAL_NAME = %Request.GetParameter("PORTAL_NAME");
&Portal.Open(PORTAL_NAME);
```


GetRemoteNodes

Syntax

`GetRemoteNodes``()`

Description

Use the GetRemoteNodes method to return a collection of nodes for the session.

Note. This method returns both local and remote nodes.

Parameters

None.

Returns

A reference to a collection of nodes if successful, Null otherwise.

Example

```
&NodeColl = %Session.GetNodes();
```

See Also

Chapter 30, "Portal Registry Classes," RemoteNode Collection, page 1652

PortalRegistry Class

A PortalRegistry object is returned from:

- The GetPortalRegistry session method.
- The PortalRegistry Collection Methods First, ItemByName, or Next.

Note. In addition to the following methods, the PortalRegistry class has methods used with the Search API.

See Also

Chapter 30, "Portal Registry Classes," GetPortalRegistry, page 1614

Chapter 30, "Portal Registry Classes," PortalRegistry Collection Methods, page 1643

Chapter 30, "Portal Registry Classes," Changing PortalRegistry Properties, page 1808

Chapter 45, "Verity Search Classes," page 2465

PortalRegistry Class Methods

In this section, we discuss the PortalRegistry class methods. The methods are discussed in alphabetical order.

Close

Syntax

Close()

Description

The Close method closes the PortalRegistry object, that is, this method sets the object to the state it was in immediately after the GetPortalRegistry was done on the PeopleSoft Session object. Any unsaved changes are discarded. The Close method can be used only on an open PortalRegistry, not a closed one. This means you must have opened the PortalRegistry with the Open method before you can close it.

Parameters

None.

Returns

A Boolean value: True if the PortalRegistry object is successfully closed, False otherwise.

Example

```
&Rslt = &MyRegistry.Close();

If Not &Rslt Then
   /* registry not closed - do error processing */
End-if;
```


See Also

Chapter 30, "Portal Registry Classes," Open, page 1634 and Chapter 30, "Portal Registry Classes," Save, page 1638

CopyObject

Syntax

CopyObject*(sourcePortalName,sourceRefType,sourceObjName,targetPortalName, targetPrntFldrName,copyChildren)*

Description

Use the CopyObject method to copy folders and content references between portal registries.

If the object being copied has related HTML, the HTML is also copied and named according to the current portal naming convention for HTML objects, which is:

PR_<*portalname*>_<*objectname*>

where <*portalname*> is up to the first 8 characters of the portal name and <*objectname*> is up to the first 17 characters of the object name.

Considerations Using CopyObject

The source portal name and the target portal name cannot be the same name.

The *targetPrntFldrName* must be a folder that exists in the portal identified by the target portal name. It cannot be null (or empty). If you need to copy an entire portal, the portal must first be "created" using the Create method, before copying over all the folders and content references.

The *copyChildren* parameter is ignored for content references. However, it still must be set to "Yes" or "No".

Parameters

Parameter	*Description*
sourcePortalName	Specify the name of the portal from which the object is to be copied.
sourceRefType	Specify the type of object to be copied. Values are: • "C" for content references • "F" for folders This parameter is case-sensitive.
sourceObjName	Specify the name of the object to be copied.

Parameter	Description
targetPortalName	Specify the name of the portal to which the object will be copied.
targetPrntFldrName	Specify the name of the folder that will be the parent folder for the source object.
copyChildren	Specify whether to also copy the children of the source object. This parameter is valid only with folder objects. Values are: • "Yes" copy child objects • "No" do not copy child objects This parameter is case-sensitive.

Returns

A Boolean value: True if object copied successfully, False otherwise.

Example

```
&Success = &Portal.CopyObject("PORTAL", "F", "PEOPLETOOLS_QUALITY", "BOGUS",⇒
 "PORTAL_ROOT_OBJECT", "Yes");
If (&Success = False) Then
   WinMessage("portal copy failed");
   Exit;
End-If;
```

Create

Syntax

Create*(RegistryName)*

Description

The Create method creates a new PortalRegistry in the PortalRegistry object called *RegistryName.* The specified registry must be a new registry. The Create method returns False if the registry already exists.

The new PortalRegistry is immediately committed to the database. If you change any property of a PortalRegistry after you create the object, use the Save method to commit your changes to the database.

When a registry is created, a folder called *Root* is automatically created. This is the root folder for the registry.

Note. If you're using Visual Basic, you must check that the PortalRegistry is actually created. If you use a duplicate name, a zero is returned, but no error results.

Parameters

Parameter	Description
RegistryName	The name of the registry to create. This parameter takes a string value. If you specify a registry that already exists, this method returns a False value.

Returns

A Boolean value: True if the PortalRegistry object is successfully created, False otherwise.

Example

```
&PORTAL_NAME = MY_PORTAL_RECORD.PORTALNAME;
&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Create(&PORTAL_NAME) Then;
   /* portal not created - do error processing */
End-If;
```

See Also

Chapter 30, "Portal Registry Classes," Open, page 1634; Chapter 30, "Portal Registry Classes," Save, page 1638; Chapter 30, "Portal Registry Classes," Close, page 1616 and Chapter 30, "Portal Registry Classes," Delete, page 1621

CreateContentRefLink

Syntax

CreateContentRefLink*(LinkName,LinkLabel,LinkParent,CRefPortalName, CRefObjectName)*

Description

Use the CreateContentRefLink method to create a link to any content reference in any portal in a local database.

For example, using this method, you can create a link to a content reference in the EMPLOYEE portal, to a content reference in the CUSTOMER portal, or to a content reference in the current portal.

Parameters

Parameter	*Description*
LinkName	Specify the name of the link as a string. This is the ID of the link. The name is validated as that of the content reference name. It cannot start with a number, and can not have special characters and spaces. This property is not translated.
LinkLabel	Specify the label of the link as a string. This property can be translated. This is the label that appears on the left hand navigation menu If this value is blank, a linked content reference label is displayed instead.
LinkParent	Specify the parent folder of the link, as a string.
CRefPortalName	Specify the portal name of the content reference, to which the link is pointing, as a string.
CRefObjectName	Specify the ID of the content reference to which the link is pointing, as a string.

Returns

A reference to a newly created ContentReference link object.

Example

```
Local ApiObject &Portal, &CRef, &CReflink;
&Portal = PortalOpen();
&CReflink = &Portal.CreateContentRefLink(<Unique Link Name> , <LinkLable>,⇒
 "<Link's Parent folder >", "<Cref Portal Name>", <Cref Object Name>);

/*... Use the link object */

&Portal.Close();
```

CreateRemote

Syntax

CreateRemote(*PortalName*, *RemoteNodeName*)

Description

Use the CreateRemote method to create a remote portal. Although you can have more than one portal on a node, they must all be uniquely named.

Parameters

Parameter	*Description*
PortalName	Specify the name of the new portal to create on the remote node.
RemoteNodeName	Specify the name of the remote node to create the new portal on.

Returns

A Boolean value: True if the PortalRegistry object is successfully created, False otherwise.

See Also

Chapter 30, "Portal Registry Classes," Create, page 1618 and Chapter 30, "Portal Registry Classes," Portals, page 1642

Delete

Syntax

`Delete``(RegistryName)`

Description

The Delete method deletes the PortalRegistry *from the database,* including any data and tables.

Warning! If you delete a PortalRegistry, you delete *everything.* Your entire PortalRegistry is gone, all the folders, content references, templates, and so on. Do not delete a PortalRegistry object unless you are absolutely certain that you want to.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

The Delete method can only be used with a *closed* registry, it cannot be used on an open registry. Before you use the Delete method, you must explicitly close the PortalRegistry object (with the Close method.) The Delete method returns False if you try to delete an open PortalRegistry object.

Parameters

Parameter	*Description*
RegistryName	The name of the registry to delete. This parameter takes a string value. If you specify a registry that doesn't exist, this method returns a False value.

Returns

A Boolean value: True if the PortalRegistry object is successfully deleted, False otherwise.

Example

The following example deletes all PortalRegistry objects that start with HRMS_99.

```
Local ApiObject &MySession;
Local ApiObject &MyPortal;

&MySession = %Session;

&MyPortal = &MySession.GetPortalRegistry();

&MyPortal.Delete("HRMS_99");
```

See Also

Chapter 30, "Portal Registry Classes," Close, page 1616 and Chapter 30, "Portal Registry Classes," Open, page 1634

DeleteHomepage

Syntax

DeleteHomepage()

Description

The DeleteHomepage method deletes the current user's homepage. This method is valid only after a user has opened a PortalRegistry object.

Parameters

None.

Returns

A Boolean value: True if the homepage is successfully deleted, False otherwise.

See Also

Chapter 30, "Portal Registry Classes," UserHomepage Class, page 1789

FindCRefByName

Syntax

`FindCRefByName`*`(Name)`*

Description

The FindCRefByName method returns the content reference object corresponding to *Name.* The name is a unique identifier for each content reference.

Considerations on Returned Content References

This method returns content reference objects that aren't yet valid as well as ones that are no longer valid. When you create your program, you must check for these invalid values if you don't want to use them. You can check using ValidTo, ValidFrom, or IsVisible.

This method returns content reference objects that you aren't authorized to. When you create your program, you should always check the Authorized property. This is the only property you can view from an object that you're not authorized to view.

Parameters

Parameter	*Description*
Name	A unique name within the registry that identifies the content reference. This parameter takes a string value. This parameter uses the name, not the label, of a content reference.

Returns

What this method returns depends on the condition of the content reference:

- If the content reference name is valid and the end-user has access to the content reference, a content reference object is returned.
- If the content reference is valid but the end-user doesn't have access to it a content reference object is returned, however, the only property you can access is the Authorized property.

- If you specified an invalid name this method returns NULL.

Example

The following example returns a GLOBAL_PAYROLL content reference object:

```
&CRef = &Portal.FindCRefByName("GLOBAL_PAYROLL");
```

See Also

Chapter 30, "Portal Registry Classes," Content Reference Class, page 1683

FindCRefByURL

Syntax

`FindCRefByURL`*`(URL)`*

Description

The FindCRefByURL method returns the content reference object corresponding to *URL.* The URL specified by *URL* must be an absolute URL.

Note. The portal registry API needs the current URI of the local node to work. During runtime, it gets this information from the current webserver. For Application Engine programs, there isn't a current webserver, so it has to get this information from the database.

If you have multiple content providers with the same URI, FindCRefByURL looks for the specified content reference with all those content providers.

To access pagelet categories and their associated pagelets, use the PageletCategories collection from the PortalRegistry object, not FindCRefByURL.

Considerations on Returned Content References

This method returns content reference objects that aren't yet valid as well as ones that are no longer valid. When you create your program, check for these properties (ValidTo and ValidFrom) if you don't want to use them.

This method returns content reference objects that the end-user isn't authorized to. When you create your program, always check the Authorized property. This is the only property you can view from an object that the end-user isn't authorized to view.

Parameters

Parameter	*Description*
URL	A URL that represents the content. This parameter takes a string value. This URL must be an absolute URL. This parameter is case-insensitive.

Returns

What this method returns depends on the condition of the content reference:

- If the end-user has access to the URL, a content reference object is returned.
- If the content reference is registered, but the end-user doesn't have access to it, a content reference is returned, but the only property you can access is the Authorized property.
- If a URL isn't registered or is invalid, this method returns NULL.

Example

The following example finds a content reference from a URL:

```
&UserCRef = &Portal.FindCRefByURL("http://www.PeopleSoft.Com");
```

See Also

Chapter 30, "Portal Registry Classes," GetQualifiedURL, page 1631 and Chapter 30, "Portal Registry Classes," QualifiedURL, page 1693

Chapter 30, "Portal Registry Classes," Content Reference Class, page 1683

FindCRefForURL

Syntax

FindCRefForURL*(URL)*

Description

The FindCRefForURL method returns the content reference object corresponding to *URL*. The URL specified by *URL* must be an absolute URL.

If the exact content reference is not found, this method tries to look for the content reference again, after stripping off the query portion of the URL.

If you don't want the system searched without the query string, use the FindCRefByURL method.

Note. The portal registry API needs the current URI of the local node to work. During runtime, it gets this information from the current webserver. For Application Engine programs, there isn't a current webserver, so it has to get this information from the database.

If you have multiple content providers with the same URI, FindCRefForURL looks for the specified content reference with all those content providers.

To access pagelet categories and their associated pagelets, use the PageletCategories collection from the PortalRegistry object, not FindCRefForURL.

Considerations on Returned Content References

This method returns content reference objects that aren't yet valid as well as ones that are no longer valid. When you create your program, check for these properties (ValidTo and ValidFrom) if you don't want to use them.

This method returns content reference objects that the end-user isn't authorized to. When you create your program, always check the Authorized property. This is the only property you can view from an object that the end-user isn't authorized to view.

Parameters

Parameter	*Description*
URL	A URL that represents the content. This parameter takes a string value. This URL must be an absolute URL. This parameter is case-insensitive.

Returns

What this method returns depends on the condition of the content reference:

- If the end-user has access to the URL, a content reference object is returned.
- If the content reference is registered, but the end-user doesn't have access to it, a content reference is returned, but the only property you can access is the Authorized property.
- If a URL isn't registered or is invalid, this method returns NULL.

Example

The following example finds a content reference from a URL. If the content reference isn't found from the full URL, the query string is stripped and the system searches again:

```
&UserCRef = &Portal.FindCRefForURL("http://www.peoplesoft.com/crefs/psportal⇒
/technologies/?url=http%3a%2f%2faugust2004%2fipass.html");
```


See Also

Chapter 30, "Portal Registry Classes," GetQualifiedURL, page 1631; Chapter 30, "Portal Registry Classes," QualifiedURL, page 1693 and Chapter 30, "Portal Registry Classes," FindCRefByURL, page 1624

Chapter 30, "Portal Registry Classes," Content Reference Class, page 1683

FindCRefLinkByName

Syntax

FindCRefLinkByName*(LinkName)*

Description

Use the FindCRefLinkByName method to find the existing link in the current portal. If the link is found a reference to ContentReference link object is returned. A Null value is returned if the link is not found in the database, and the error message is added to message collection

Considerations on Returned Links

This method returns links that aren't yet valid as well as ones that are no longer valid. When you create your program, you must check for these properties (ValidTo and ValidFrom) if you don't want to use them. This method may also return objects that you aren't authorized to. When you create your program, you should always check the Authorized property. This is the only property you can view from an object that you're not authorized to view.

Parameters

Parameter	*Description*
LinkName	Specify the name of the link as a string. This is the ID of the link.

Returns

A reference to an existing ContentReference link object. or a Null value if the link is not found.

FindFolderByName

Syntax

FindFolderByName*(Name)*

Description

The FindFolderByName method returns the Folder object corresponding to *Name.* The name is a unique identifier for each folder.

Considerations on Returned Folders

This method returns Folder objects that aren't yet valid as well as ones that are no longer valid. When you create your program, check for these properties (ValidTo and ValidFrom) if you don't want to use them.

This method returns folder objects that you aren't authorized to. When you create your program, always check the Authorized property. This is the only property you can view from an object that you're not authorized to view.

Parameters

Parameter	*Description*
Name	A unique name within the registry that identifies the folder. This parameter takes a string value. This parameter takes the name of a folder, not the label.

Returns

What this method returns depends on the condition of the folder:

- If the folder name is valid and the end-user has access to the folder, a folder object is returned.
- If the folder is valid but the end-user doesn't have access to it a folder object is returned, however, the only property you can access is the Authorized property.
- If you specified an invalid name this method returns NULL.

Example

The following example returns a folder named ROOT:

```
&MyFolder = &MyPortal.FindFolderByName("ROOT");
```

The following example returns the folder object for an already instantiated content reference:

```
&Folder = &Portal.FindFolderByName(&CRef.ParentName);
```

See Also

Chapter 30, "Portal Registry Classes," Folder Class, page 1663

FindPgltByName

Syntax

`FindPgltByName``(`*`PageletName`*`)`

Description

The FindPgltByName method returns the pagelet object corresponding to *PageletName.* The name is a unique identifier for each pagelet.

Considerations on Returned Pagelets

This method returns pagelet objects that aren't yet valid as well as ones that are no longer valid. When you create your program, check for these properties (ValidTo and ValidFrom) if you don't want to use them.

This method returns pagelet objects that you aren't authorized to. When you use create your program, always check the Authorized property. This is the only property you can view from an object that you're not authorized to view.

Parameters

Parameter	***Description***
PageletName	A unique name within the registry that identifies the Pagelet. This parameter takes a string value. This parameter takes the name of a Pagelet, not the label.

Returns

What this method returns depends on the condition of the Pagelet:

- If the Pagelet name is valid and the end-user has access to the Pagelet, a Pagelet object is returned.
- If the Pagelet is valid but the end-user doesn't have access to it a Pagelet object is returned, however, the only property you can access is the Authorized property.
- If you specified an invalid name this method returns NULL.

Example

The following example returns a pagelet named HomePg_Dictionary:

```
&MyPglt = &MyPortal.FindPgltByName("HomePg_Dictionary");
```

See Also

Chapter 30, "Portal Registry Classes," Pagelet Class, page 1777

GetAbsoluteContentURL

Syntax

`GetAbsoluteContentURL``(`*`NodeName,URL`*`)`

Description

The GetAbsoluteContentURL method returns the absolute unwrapped simple URL of the content, in the context of the current portal. For example, if the PortalRegistry object accessed a portal called Employees, and the method were called like this:

```
&Registry.getAbsoluteContentURL(Node.CRM,  "/c/SERVICES.ORDERS.GBL");
```

It would return the following string:

```
http://crmserver/servlets/psc/crmHome/Employees/CRM/c/SERVICES.ORDERS.GBL
```

In the returned string, the portion of the string from server name through *portal_home* (crmHome in the example) are the ones associated with the content node (CRM in the example). The portal is the current portal, and the rest of the URL is the URL string passed in.

Parameters

Parameter	***Description***
NodeName	Specify the name of the node that contains the content. You can also use a string, such as %Node, for this value.
URL	Specify the relative URL pointing to the content that you want the absolute URL generated for.

Returns

A string containing the absolute URL.

GetDefaultHPTabOID

Syntax

`GetDefaultHPTabOID``()`

Description

Use the GetDefaultHPTabOID method to return the name of the first homepage tab that is found and authorized for the current user ID.

The search order of the tabs depends on Sequence number of the tab name and is sorted alphabetically.

Parameters

None.

Returns

A string

GetQualifiedURL

Syntax

GetQualifiedURL*(ContentProvider, RelativeURL)*

Description

Note. This method is maintained only for backward compatibility. If your existing code uses this method, it actually returns the value from GetAbsoluteContentURL method. New applications should use the GetAbsoluteContentURL method.

See Also

Chapter 30, "Portal Registry Classes," GetAbsoluteContentURL, page 1630

GrantPermissionForComponent

Syntax

GrantPermissionForComponent*(MenuName,ComponentName, Market,PermListName, NodeName)*

Description

Use the GrantPermissionForComponent method to grant the specified permission and cascaded upwards on parent folders to the specified component. In addition, the specified permission is granted to any component references that point to the specified component, and any parent folders.

Components that have query strings are also searched and permissions are applied on them.

If you use the string "LOCAL_NODE" as *NodeName,* the system uses the node name currently defined as local.

All component entries in the portal registry are affected for all the portals. Not just the current portal.

Parameters

Parameter	***Description***
MenuName	Specify the menu name, as a string, that the component you want to grant permissions for is associated with.
ComponentName	Specify the component name, as a string, that you want to grant permissions for.
Market	Specify the market, as a string, associated with the component that you want to grant permissions for.
PermListName	Specify the name of the permission list, as a string, that you want to use.
NodeName	Specify the node of the component reference, as a string, that points to the component reference that you want to grant permissions for. If you use the string "LOCAL_NODE", the system uses the node name currently defined as local.

Returns

A Boolean value: True if method completed successfully, False otherwise.

Example

The My Profile Component Reference points to the USERMAINT_SELF component. Using the following example, "ALLPANLS" permission is granted to the MY_PROFILE component reference, as well as "MY INFO", the parent folder.

```
&Portal.GrantPermissionForComponent("MAINTAIN_SECURITY", "USERMAINT_SELF", "GBL",⇒
 "ALLPANLS", "LOCAL_NODE");
```

See Also

Chapter 30, "Portal Registry Classes," RevokePermissionForComponent, page 1636

PeopleTools 8.51 PeopleBook: Security Administration, "Setting Up Permission Lists"

GrantPermissionForScript

Syntax

`GrantPermissionForScript`*`(RecordName,FieldName,EventName,FuncName,PermListName, NodeName)`*

Description

Use the GrantPermissionForScript method to grant the specified permission to the specified iScript. In addition, the specified permission is granted to any component references that point to the specified iScript, and any parent folders.

If you use the string "LOCAL_NODE" as *NodeName,* the system uses the node name currently defined as local.

All iScript entries in the portal registry are affected for all the portals. Not just the current portal.

Parameters

Parameter	***Description***
RecordName	Specify the record name, as a string, of the record containing the iScript. All iScripts are contained on records whose names start with "WEBLIB".
FieldName	Specify the name of the field, as a string, containing the iScript.
EventName	Specify the name of the event, as a string, containing the iScript. Generally iScripts are contained in the FieldFormula event.
FuncName	Specify the name of the function, as a string, containing the iScript.
PermListName	Specify the name of the permission list, as a string, that you want to use.
NodeName	Specify the node of the component reference, as a string, that points to the component reference that you want to grant permissions for. If you use the string "LOCAL_NODE", the system uses the node name currently defined as local.

Returns

A Boolean value: True if method completed successfully, False otherwise.

Example

```
&Portal.GrantPermissionForScript("WEBLIB_ALERT", "ALERTCOUNT", "FieldFormula",⇒
 "Connect_Alert", "ALLPGS", "Local_Node");
```


See Also

Chapter 30, "Portal Registry Classes," RevokePermissionForComponent, page 1636

PeopleTools 8.51 PeopleBook: Security Administration, "Setting Up Permission Lists"

Open

Syntax

Open*(RegistryName)*

Description

The Open method opens the PortalRegistry specified by the parameters. The registry must already exist. The Open method can be used only with a *closed* PortalRegistry, it cannot be used on an open registry.

Parameters

Parameter	***Description***
RegistryName	The name of the registry to open. This parameter takes a string value. If you specify a registry that doesn't exist, this method returns a False value.

Returns

A Boolean value: True if the PortalRegistry object is successfully opened, False otherwise.

Example

In the following example, the name of the portal is stored as the value of a field in a record.

```
&PORTAL_NAME = EO_PE_REG_AET.PORTAL_NAME;
&Portal = %Session.GetPortalRegistry();

&Portal.Open(&PORTAL_NAME);
```

See Also

Chapter 30, "Portal Registry Classes," Open, page 1634; Chapter 30, "Portal Registry Classes," Save, page 1638; Chapter 30, "Portal Registry Classes," Close, page 1616 and Chapter 30, "Portal Registry Classes," Delete, page 1621

PermissionListDelete

Syntax

PermissionListDelete*(PermListName)*

Description

Use the PermissionListDelete method to delete the specified permission list from the PortalRegistry.

If you try to delete a permission list that is still in use, you receive an error when you try to save the object.

The permission list is deleted from all the portal objects for all the portal.

Parameters

Parameter	***Description***
PermListName	Specify the permission list to delete from the PortalRegistry.

Returns

A Boolean value: True if the permission list is successfully deleted, False otherwise.

PermissionListSaveAs

Syntax

PermissionListSaveAs*(PermListSourceName,PermListTargetName*)

Description

Use the PermissionListSaveAs method to copy the specified permission list in the PortalRegistry.

Parameters

Parameter	***Description***
PermListSourceName	Specify the name of the PermissionList to copy.
PermListTargetName	Specify the name that you want to give the new PermissionList.

Returns

A Boolean value: True if the permission list is successfully copied, False otherwise.

RevokePermissionForComponent

Syntax

RevokePermissionForComponent*(MenuName,ComponentName, Market,PermListName, NodeName)*

Description

Use the RevokePermissionForComponent method to revoke the specified permission to the specified component. In addition, the specified permission is revoked for any component references that point to the specified component.

If you use the string "LOCAL_NODE" as *NodeName,* the system uses the node name currently defined as local.

All component entries in the portal registry are affected for all the portals. Not just the current portal.

Parameters

Parameter	***Description***
MenuName	Specify the menu name, as a string, that the component that you want to revoke permissions for is associated with.
ComponentName	Specify the component name, as a string, that you want to revoke permissions for.
Market	Specify the market, as a string, associated with the component that you want to revoke permissions for.
PermListName	Specify the name of the permission list, as a string, that you want to use.
NodeName	Specify the node of the component reference, as a string, that points to the component reference that you want to revoke permissions for. If you use the string "LOCAL_NODE", the system uses the node name currently defined as local.

Returns

A Boolean value: True if method completed successfully, False otherwise.

Example

The My Profile Component Reference points to the USERMAINT_SELF component. Using the following example, "ALLPANLS" permission is revoked to the MY_PROFILE component reference.

```
&Portal.RevokePermissionForComponent("MAINTAIN_SECURITY", "USERMAINT_SELF", "GBL",⇒
 "ALLPANLS", "LOCAL_NODE");
```

See Also

Chapter 30, "Portal Registry Classes," GrantPermissionForComponent, page 1631

PeopleTools 8.51 PeopleBook: Security Administration, "Setting Up Permission Lists"

RevokePermissionForScript

Syntax

`RevokePermissionForScript`*`(RecordName,FieldName, EventName,FuncName,PermListName, NodeName)`*

Description

Use the RevokePermissionForScript method to revoke the specified permission to the specified iScript. In addition, the specified permission is revoked for any component references that point to the specified iScript.

If you use the string "LOCAL_NODE" as *NodeName,* the system uses the node name currently defined as local.

All component entries in the portal registry are affected for all the portals. Not just the current portal.

Parameters

Parameter	***Description***
RecordName	Specify the record name, as a string, of the record containing the iScript. All iScripts are contained on records whose names start with "WEBLIB".
FieldName	Specify the name of the field, as a string, containing the iScript.
EventName	Specify the name of the event, as a string, containing the iScript.
FuncName	Specify the name of the function, as a string, containing the iScript. Generally iScripts are contained in the FieldFormula event.
PermListName	Specify the name of the permission list, as a string, that you want to use.

Parameter	Description
NodeName	Specify the node of the component reference, as a string, that points to the component reference that you want to revoke permissions for. If you use the string "LOCAL_NODE", the system uses the node name currently defined as local.

Returns

A Boolean value: True if method completed successfully, False otherwise.

Example

```
&Portal.RevokePermissionForScript("WEBLIB_ALERT", "ALERTCOUNT", "FieldFormula",⇒
 "Connect_Alert", "ALLPGS", "Local_Node");
```

See Also

Chapter 30, "Portal Registry Classes," GrantPermissionForScript, page 1633

PeopleTools 8.51 PeopleBook: Security Administration, "Setting Up Permission Lists"

Save

Syntax

```
Save()
```

Description

The Save method saves changes that you made to the PortalRegistry or to a node template. It does *not* save any changes that you made to a folder, content reference, and so on.

Note. To save changes for a folder or content reference use the save method associated with that object.

Parameters

None.

Returns

A Boolean value: True if the PortalRegistry object is successfully saved, False otherwise.

Example

```
If Not(&MyPortal.Save()) Then
   /* do error checking */
End-if:
```

See Also

Chapter 30, "Portal Registry Classes," Save, page 1663 folder method

Chapter 30, "Portal Registry Classes," Save, page 1684 content reference method

PortalRegistry Class Properties

In this section, we discuss the PortalRegistry class properties. The properties are discussed in alphabetical order.

DefaultTemplate

Description

This property returns or sets the name of the default template for this PortalRegistry object as a string.

If you delete a template for a content reference, and none of the other content references on a page have a template, the default template specified with the Node is used. If there's no template for the Node, the template specified with this property is used.

To return a reference to the content reference that contains the template specified by this property, use the TemplateObject property.

Note. If you change the Default Template for a portal, the template won't take effect until you close all the existing browser windows with the session and open a new browser window.

This property takes only the first 30 characters of a value. If you specify a value longer than 30 characters, the remaining characters are ignored.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," TemplateObject, page 1643

Description

Description

This property returns or sets the description of this PortalRegistry object as a string

The length of this property is 256 characters.

This property is read-write.

Favorites

Description

This property returns a reference to the Favorites Collection for the current end-user.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Favorite Collection, page 1804

FolderNavObject

Description

This property specifies the content reference name to be used for the folder navigation object. When the folder navigation is tunred on, this content reference is displayed as the folder navigation homepage. This property takes a string value.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," IsFolderNavigation, page 1641

Homepage

Description

This property returns a reference to the Homepage for the current end-user.

This property is read-only.

IsFolderNavigation

Description

This property specifies if there is folder navigation. When the folder navigation is turned on, the user can see a folder homepage when clicked on the folder in the lefthand navigation menu. This property takes a Boolean value: true if folder navigation is on, false otherwise.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," FolderNavObject, page 1640

Name

Description

This property returns the name of the PortalRegistry object as a string.

The length of this property is 30 characters.

This property is read-only.

NodeTemplates

Description

This property returns a reference to a NodeTemplate Collection. This property can be used with a *closed* portal registry, that is, before you open it with the Open method.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," NodeTemplate Collection, page 1659

OwnerId

Description

This property returns or sets the owner ID of the PortalRegistry object as a string.

This property is read-write.

PageletCategories

Description

This property returns a reference to a PageletCategories Collection that contains all the PageletCategories for a portal.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," PageletCategory Collection, page 1773

Portals

Description

This property returns a reference to a Portal collection that contains references to all the portals in the database.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Portal Collection, page 1656

RootFolder

Description

This property returns the root folder object for this PortalRegistry object.

This property is read-only.

TabDefinitions

Description

This property returns reference to a TabDefinitions Collection that contains all the TabDefinitions for a portal.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," TabDefinition Collection, page 1745

TemplateObject

Description

This property returns a reference to a content reference object that contains the template specified by the DefaultTemplate property. If no template is specified with DefaultTemplate, this property returns Null.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," DefaultTemplate, page 1639

PortalRegistry Collection

A PortalRegistry Collection is returned by the FindPortalRegistries session class method.

See Chapter 30, "Portal Registry Classes," FindPortalRegistries, page 1611.

PortalRegistry Collection Methods

In this section, we discuss the PortalRegistry collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first PortalRegistry object in the PortalRegistry collection.

Example

```
&MyRegistry = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the PortalRegistry object with the position in the PortalRegistry collection specified by *number*.

Parameters

Parameter	*Description*
number	Specify the position number in the collection of the PortalRegistry object that you want returned.

Returns

A PortalRegistry object if successful, NULL otherwise.

Example

```
For &I = 1 to &MyCollection.Count
   &MyRegistry = &MyCollection.Item(&I);
   /* Do processing */
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next PortalRegistry object in the PortalRegistry collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MyRegistry = &MyCollection.Next();
```

PortalRegistry Collection Property

This section discusses the Count property.

Count

Description

This property returns the number of PortalRegistry objects in the PortalRegistry collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Node Class

The node indicates the URI of the content server. If the node is a PeopleSoft content server, it has a PeopleSoft server URI. If the node has external content, the URI indicates the external content.

Use the IsLocal or IsDefault to determine if a node from a RemoteNode collection is remote or local.

Node objects are instantiated from the following:

- From a Session object with the GetLocalNode method.
- From a Node Collection with the First, InsertItem, ItemByName, and Next methods.
- From a RemoteNode Collection with the First, InsertItem, ItemByName, and Next methods.

See Chapter 30, "Portal Registry Classes," GetLocalNode, page 1613; Chapter 30, "Portal Registry Classes," Node Collection, page 1649 and Chapter 30, "Portal Registry Classes," RemoteNode Collection, page 1652.

Node Class Properties

In this section, we discuss the Node class properties. The properties are discussed in alphabetical order.

ActiveNode

Description

This property indicates whether the node has been specified as an active node. This property returns a Boolean value: True, the node is active, False otherwise.

This property is read-only.

AppsRelease

Description

This property returns the release of the PeopleSoft Applications hosted on this node as a string. This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," NodeType, page 1648

ContentURI

Description

This property returns the URI for the content webserver of this node as a string.

This property is read-only.

DefaultPortalName

Description

This property returns the name of the portal associated with the node as the default portal for the node.

This property is read-only.

Description

Description

This property returns the description for this node as a string.

The length of this property depends on your system database limit for LONG fields.

This property is read-only.

IsDefault

Description

This property indicates whether the node has been specified as the default local node. This property takes a Boolean value: True, this node has been specified as the default local node, False otherwise.

This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," NodeType, page 1648

IsLocal

Description

This property indicates whether this node has been specified as the local PeopleSoft node. This property takes a Boolean value: True, this node has been specified as a local node, False otherwise.

This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," NodeType, page 1648

Name

Description

This property returns the name for this node object as a string. The name is a unique identifier for each node object.

Every node name must be unique in the database.

This property is read-only.

NodePassword

Description

This property returns the password for this node object as a string.

This property is read-only.

NodeType

Description

This property indicates what the node is used for. Nodes can either define PeopleSoft or external systems. PeopleSoft nodes have more capabilities than external nodes, therefore several other properties depend on this property.

Values are:

Value	***Description***
PIA	PIA (Peoplesoft 8.4 simple URL format)
EX	External Node
ICT	ICType (used for 8.1x content)

PIA is the default value on a new Node.

This property is read-only.

PortalURI

Description

This property returns the URI for the portal webserver of this node as a string. This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," NodeType, page 1648

ToolsRelease

Description

This property returns the release of PeopleTools running on this node as a string. This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," NodeType, page 1648

Node Collection

A node collection contains a set of references to all nodes defined in the database.

Node Collection Methods

In this section, we discuss the Node collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```


Description

The First method returns the first Node object in the Node collection.

Parameters

None.

Returns

A Node object.

Example

```
&MyNode = &MyCollection.First();
```

ItemByName

Syntax

`ItemByName`*`(NodeName)`*

Description

The ItemByName method returns the Node object with the name specified by *NodeName.*

Parameters

Parameter	***Description***
NodeName	Specify the name of an existing node in the Node collection. If you specify an invalid name, the object returned is NULL.

Returns

A Node object if successful, NULL otherwise.

Example

```
&MyOldNode = &MyPortal.Nodes.ItemByName("HRMS");
```


Next

Syntax

Next()

Description

The Next method returns the next Node object in the Node collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A Node object.

Example

```
&MyNode = &MyCollection.Next();
```

Node Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Node objects in the Node Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```


RemoteNode Collection

A remote node collection contains a set of references to all the nodes defined in the database, both local and remote.

Use the IsLocal property of the returned node object to determine if the node is remote or local.

RemoteNode Collection Methods

In this section, we discuss the RemoteNode collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first Node object in the RemoteNode collection.

Parameters

None.

Returns

A Node object.

Example

```
&MyNode = &MyCollection.First();
```

ItemByName

Syntax

ItemByName*(NodeName)*

Description

The ItemByName method returns the Node object with the name specified by *NodeName*.

Parameters

Parameter	***Description***
NodeName	Specify the name of an existing node in the RemoteNode collection. If you specify an invalid name, the object returned is NULL.

Returns

A Node object if successful, NULL otherwise.

Example

```
&MyOldNode = &MyPortal.RemoteNodes.ItemByName("HRMS");
```

Next

Syntax

```
Next()
```

Description

The Next method returns the next Node object in the RemoteNode collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A Node object.

Example

```
&MyNode = &MyCollection.Next();
```

RemoteNode Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Node objects in the RemoteNode Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Portal Class

The portal class provides access to portals declared in the session.

A portal object is instantiated by the First, InsertItem, ItemByName, and Next Portal Collection methods

See Chapter 30, "Portal Registry Classes," Portal Collection, page 1656.

Portal Class Method

In this section, we discuss the Portal class methods. The methods are discussed in alphabetical order.

Save

Syntax

Save()

Description

Use the Save method to save changes you made to the portal object.

Parameters

None.

Returns

A Boolean value: True if the portal object is successfully saved, False otherwise.

See Also

Chapter 30, "Portal Registry Classes," Save, page 1638 PortalRegistry method

Portal Class Properties

In this section, we discuss the Portal class properties. The properties are discussed in alphabetical order.

HostNodeName

Description

This property sets or returns the name of the node hosting the portal as a string.

This property is read-write.

IsLocal

Description

This property indicates whether the portal is defined as local or remote. This property returns a Boolean value: True, the portal is defined as local, False otherwise.

This property is read-only.

Name

Description

This property returns the name of this portal as a string. This name exactly matches the portal name of the portal registry defined in the database.

This property is read-only.

Portal Collection

The portal collection contains a set of references to each portal defined in the database.

A portal collection is instantiated by the Portals PortalRegistry property.

See Chapter 30, "Portal Registry Classes," Portals, page 1642.

Portal Collection Methods

In this section, we discuss the Portal collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first Portal object in the Portal collection.

Parameters

None.

Returns

A Portal object.

Example

```
&MyPortal = &MyCollection.First();
```

ItemByName

Syntax

ItemByName*(PortalName)*

Description

The ItemByName method returns the Portal object with the name *PortalName*.

Parameters

Parameter	*Description*
PortalName	Specify the name of an existing portal within the collection. If you specify an invalid name, the object returns NULL.

Returns

A Portal object if successful, NULL otherwise.

Example

```
&MyPortal = &MyPortal.Portals.ItemByName("HRMS");
```

Next

Syntax

Next()

Description

The Next method returns the next Portal object in the Portal collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A Portal object.

Example

```
&MyPortal = &MyCollection.Next();
```

Portal Collection Properties

In this section, we discuss the Count property.

Count

Description

This property returns the number of Portal objects in the Portal Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

NodeTemplate Class

The NodeTemplate class provides access to the default template assigned to a node in this portal.

A NodeTemplate object is instantiated by the First, InsertItem, ItemByName, and Next NodeTemplate Collection methods.

See Chapter 30, "Portal Registry Classes," NodeTemplate Collection, page 1659.

NodeTemplate Class Properties

In this section, we discuss the NodeTemplate class properties. The properties are discussed in alphabetical order.

DefaultTemplate

Description

This property specifies the name of the template to be applied to the node.

This property is read-write.

Name

Description

This property specifies the name of node that the template is to be applied to.

This property is read-write.

TemplateObject

Description

This property returns a reference to the template object associated with this node as a content reference.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Content Reference Class, page 1683

NodeTemplate Collection

The NodeTemplate collection contains a set of references to each node template object in a portal.

A NodeTemplate collection is instantiated by the NodeTemplates PortalRegistry property.

See Chapter 30, "Portal Registry Classes," NodeTemplates, page 1641.

NodeTemplate Collection Methods

In this section, we discuss the NodeTemplate collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

```
DeleteItem(NodeName)
```


Description

The DeleteItem method deletes the NodeTemplate object identified by *NodeName* from the NodeTemplate Collection.

This method is not executed automatically. It is executed only when the PortalRegistry is saved.

Parameters

Parameter	***Description***
NodeName	Specify the name of a NodeTemplate to delete.

Returns

A Boolean value: True if the NodeTemplate was deleted, False otherwise.

Example

```
If Not &MyNodeTemplates.DeleteItem("HRMS") Then
   /* Do error processing */
End-if
```

First

Syntax

First()

Description

The First method returns the first NodeTemplate object in the NodeTemplate collection.

Parameters

None.

Returns

A NodeTemplate object.

Example

```
&MyNodeTemplate = &MyCollection.First();
```

InsertItem

Syntax

InsertItem*(NodeName)*

Description

The InsertItem method inserts the NodeTemplate object identified by *NodeName* into the NodeTemplate Collection.

This method is not executed automatically. It is executed only when the PortalRegistry is saved.

Parameters

Parameter	***Description***
NodeName	Specify the name of the node template to insert.

Returns

A reference to the new NodeTemplate object if the method executed successfully, NULL otherwise.

Example

```
&NewNodeTemplate = &MyPortal.NodeTemplates.InsertItem("CRM");
```

ItemByName

Syntax

ItemByName*(NodeName)*

Description

The ItemByName method returns the NodeTemplate object with the name *NodeName.*

Parameters

Parameter	*Description*
NodeName	Specify the name of an existing NodeTemplate object in the collection. If you specify an invalid name, the object is NULL.

Returns

A NodeTemplate object if successful, NULL otherwise.

Example

```
&MyNodeTemplate = &MyPortal.NodeTemplates.ItemByName("HRMS");
```

Next

Syntax

Next()

Description

The Next method returns the next NodeTemplate object in the NodeTemplate collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A NodeTemplate object.

Example

```
&MyNodeTemplate = &MyCollection.Next();
```

NodeTemplate Collection Properties

In this section, we discuss the Count property.

Count

Description

This property returns the number of NodeTemplate objects in the NodeTemplate Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Folder Class

Folder objects are instantiated from other classes as follows:

- From a PortalRegistry object with the RootFolder property or the FindFolderByName method.
- From a Folder Collection with the First, ItemByName, or Next methods

See Chapter 30, "Portal Registry Classes," RootFolder, page 1642; Chapter 30, "Portal Registry Classes," FindFolderByName, page 1627 and Chapter 30, "Portal Registry Classes," Folder Collection, page 1678.

See Chapter 30, "Portal Registry Classes," Adding a Folder, page 1811.

Folder Class Method

In this section, we discuss the Folder class methods. The methods are discussed in alphabetical order.

Save

Syntax

```
Save()
```

Description

The Save method saves any changes you made to the folder, for example, a changed description or ValidFrom date.

Using this method also saves any changes you made to PermissionValue (both role-based and non role-based) and AttributeValue objects associated with this folder. Security permissions added are cascade upward until the root folder or first public folder. Removed security permissions are removed from all the parent folders until root or public folder in folder hierarchy.

Parameters

None.

Returns

A Boolean value: True if the Folder object is successfully saved, False otherwise.

Example

```
If Not(&MyFolder.Save()) Then
   /* do error checking */
End-If;
```

See Also

Chapter 30, "Portal Registry Classes," Save, page 1638 PortalRegistry method

Chapter 30, "Portal Registry Classes," Save, page 1684 content reference method

Folder Class Properties

In this section, we discuss the Folder class properties. The properties are discussed in alphabetical order.

AbnContentProvider

Description

Use this property to set or return a string representing the node on which the SmartNavigation content resides.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," AbnDataSource, page 1665; Chapter 30, "Portal Registry Classes," AbnPeopleCode, page 1665; Chapter 30, "Portal Registry Classes," DefaultChartNavigation, page 1669; Chapter 30, "Portal Registry Classes," TreeEffectiveDate, page 1673; Chapter 30, "Portal Registry Classes," TreeName, page 1674; Chapter 30, "Portal Registry Classes," TreeSetId, page 1675; Chapter 30, "Portal Registry Classes," TreeStructureName, page 1676 and Chapter 30, "Portal Registry Classes," TreeUserKeyValue, page 1677

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

AbnDataSource

Description

Use this property to set or return a string indicating whether the SmartNavigation data source is a tree ("T") or a rowset ("R").

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," AbnContentProvider, page 1664; Chapter 30, "Portal Registry Classes," AbnPeopleCode, page 1665; Chapter 30, "Portal Registry Classes," DefaultChartNavigation, page 1669 ; Chapter 30, "Portal Registry Classes," TreeEffectiveDate, page 1673; Chapter 30, "Portal Registry Classes," TreeName, page 1674; Chapter 30, "Portal Registry Classes," TreeSetId, page 1675; Chapter 30, "Portal Registry Classes," TreeStructureName, page 1676 and Chapter 30, "Portal Registry Classes," TreeUserKeyValue, page 1677

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

AbnPeopleCode

Description

Use this property to set or return a string representing the path and PeopleCode object to execute when processing this SmartNavigation folder.

For example, if the PeopleCode program to execute is a method, then AbnPeopleCode property would be constructed from these elements:

```
APP_PACKAGE_NAME.Class_Path.Method
```

For example, if the PeopleCode program to execute is an iScript, then AbnPeopleCode property would be constructed from these elements:

```
RECORD_NAME.FIELD_NAME.PC_Event.Function
```

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," AbnContentProvider, page 1664; Chapter 30, "Portal Registry Classes," AbnDataSource, page 1665; Chapter 30, "Portal Registry Classes," DefaultChartNavigation, page 1669; Chapter 30, "Portal Registry Classes," TreeEffectiveDate, page 1673; Chapter 30, "Portal Registry Classes," TreeName, page 1674; Chapter 30, "Portal Registry Classes," TreeSetId, page 1675; Chapter 30, "Portal Registry Classes," TreeStructureName, page 1676 and Chapter 30, "Portal Registry Classes," TreeUserKeyValue, page 1677

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

Attributes

Description

This property returns an Attribute Collection containing the AttributeValue objects for this folder.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Attribute Collection, page 1707

Author

Description

This property returns the author (PeopleSoft user ID) for this folder as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the folder has access to the folder. This property takes a Boolean value. The default value for this property for a newly created object is True. This property is not cascaded.

This property is read-write.

Authorized

Description

This property specifies whether the user is authorized to view this Folder.

This property is used when you access a particular Folder using FindFolderByName. If you've specified a valid Folder with this method, a Folder is always returned, whether you are authorized to view it or not. This is the only property you can view from an object that you are not authorized to.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this folder.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," FindFolderByName, page 1627

CascadedPermissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for all the child and parent objects (up to the root folder). To determine only the permissions of the object use the Permission property instead. To determine the role-based permissions of the object use the CascadedRolePermissions property instead.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions or RolePermissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1668; Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712 and Chapter 30, "Portal Registry Classes," Permissions, page 1672

CascadedRolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for all the child and parent objects (up to the root folder). To determine only the role-based permissions of the object use the RolePermission property instead. To determine the non role-based permissions of the object use the CascadedPermissions property.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedRolePermissions property. You can add values only to the collection returned by the Permissions or RolePermissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1667; Chapter 30, "Portal Registry Classes," RolePermissions, page 1673 and Chapter 30, "Portal Registry Classes," RolePermissionValue Collection, page 1717

ContentRefs

Description

This property returns the ContentReference Collection for this folder.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Content Reference Collection, page 1700

CreationDate

Description

This property returns the creation date for this folder as a string.

This property is read-only.

DefaultChartNavigation

Description

Use this property to set or return a Boolean value indicating whether the default PeopleTools chart navigation page (PT_ABN_ORGCHART) is to be used for this folder: True, use the default PeopleTools chart navigation page; False, use the folder specified by the value Folder Navigation Object Name field (from the Folder Administration page) as the navigation object.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," AbnContentProvider, page 1664; Chapter 30, "Portal Registry Classes," AbnDataSource, page 1665; Chapter 30, "Portal Registry Classes," AbnPeopleCode, page 1665; Chapter 30, "Portal Registry Classes," TreeEffectiveDate, page 1673; Chapter 30, "Portal Registry Classes," TreeName, page 1674; Chapter 30, "Portal Registry Classes," TreeSetId, page 1675; Chapter 30, "Portal Registry Classes," TreeStructureName, page 1676 and Chapter 30, "Portal Registry Classes," TreeUserKeyValue, page 1677

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

Description

Description

This property returns or sets the description for this folder as a string.

The length of this property is 256 characters. This property is translatable.

This property is read-write.

Folders

Description

This property returns a reference to the Folder Collection for this folder.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Folder Collection, page 1678

IsMobile

Description

Note. PeopleSoft Mobile Agent is a deprecated product. This mobile property currently exists for backward compatibility only.

This property returns True if the folder is used with mobile applications, False otherwise.

This property is read-write.

IsVisible

Description

This property returns True if the Hide from Portal Navigation check box is *not* selected when the folder is created. If the folder is hidden from portal navigation, this property returns False.

Considerations Using IsVisible

If you do not specifically set this property on a new or copied folder, the IsVisible property is set as follows:

- If the ValidFrom date is less then or equal to that day's date, the IsVisible property is set to True.
- If there is no ValidTo date the IsVisible property is set to True.
- If the ValidTo date is greater then or equal to that day's date, the IsVisible property is set to True.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," ValidFrom, page 1678 and Chapter 30, "Portal Registry Classes," ValidTo, page 1678

Label

Description

This property returns or sets the label for this folder as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

Name

Description

This property returns the name for this folder as a sting. The name is a unique identifier for each folder.

Every folder name must be unique across the portal, not just in the parent folder.

This property is *not* translatable. However, the values for the Label and Description properties are translatable.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Description, page 1669 and Chapter 30, "Portal Registry Classes," Label, page 1670

OwnerId

Description

This property returns the name of the owner for this folder as a string.

This property is read-write.

ParentName

Description

This property returns the parent folder name for this folder as a string. This property is valid only if the folder is contained within another folder. If the folder using this property is the root folder, this property returns an empty string.

This property is read-only.

Path

Description

The Path property returns a path to this folder, with each element of the path separated by a period. The path has the following syntax:

```
FolderLabel{Name}.[ChildFolderLabel{Name}.]. . .
```


This property is read-only.

Example

```
Departments{PORTA_ROOT_OBJECT}.HR{EastCoast}.AdministerWorkforce{Global}
```

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for this folder. To determine the permissions for all the parent objects (up to the root folder) use the CascadedPermissions property. To access the role-based permissions, use the RolePermissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712; Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1667 and Chapter 30, "Portal Registry Classes," RolePermissionValue Collection, page 1717

Product

Description

This property returns or sets the PeopleSoft product for this folder as a string.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property indicates whether a folder is generally accessible, that is, if this property is set to True, any user can access the folder. This property is not cascaded.

This property takes a Boolean value.

The default value for this property for a newly created object is False.

This property is read-write.

RolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for this folder. To determine the role-based permissions for all the parent objects (up to the root folder) use the CascadedRolePermissions property. To access the non role-based permissions, use the Permissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1668; Chapter 30, "Portal Registry Classes," Permissions, page 1672 and Chapter 30, "Portal Registry Classes," RolePermissionValue Collection, page 1717

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned folders is based on the sequence number. Use this property to reorder folders.

If there are duplicates in the sequence number, the folders are returned alphabetically.

The length of this property is 4 characters.

This property is read-write.

TreeEffectiveDate

Description

Use this property to set or return the effective date for the tree to be used as the SmartNavigation data source as a string.

Note. If the SmartNavigation data source is a rowset, this property is set to the Null string.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," AbnContentProvider, page 1664; Chapter 30, "Portal Registry Classes," AbnDataSource, page 1665; Chapter 30, "Portal Registry Classes," AbnPeopleCode, page 1665; Chapter 30, "Portal Registry Classes," DefaultChartNavigation, page 1669; Chapter 30, "Portal Registry Classes," TreeName, page 1674; Chapter 30, "Portal Registry Classes," TreeSetId, page 1675; Chapter 30, "Portal Registry Classes," TreeStructureName, page 1676 and Chapter 30, "Portal Registry Classes," TreeUserKeyValue, page 1677

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

TreeName

Description

Use this property to set or return the name for the tree to be used as the SmartNavigation data source as a string.

Note. If the SmartNavigation data source is a rowset, this property is set to the Null string.

SmartNavigation passes the values of several tree-specific fields to the application via URL. Certain characters are inappropriate for use in a URL and must be avoided. When using a tree as a SmartNavigation data source, do not use any of the following characters in the tree name, setID, user key value, and tree branch values:

pound (#)	percent (%)	dollar ($)
ampersand (&)	plus (+)	comma (,)
forward slash/virgule (/)	colon (:)	semi-colon (;)
equals (=)	question mark (?)	at symbol (@)
space ( )	quotation marks(")	less than symbol (<)
greater than symbol (>)	left curly brace ({)	right curly brace (})
vertical bar/pipe (|)	backslash (\)	caret (^)
tilde (~)	left square bracket ([)	right square bracket (])
grave accent (`)		

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," AbnContentProvider, page 1664; Chapter 30, "Portal Registry Classes," AbnDataSource, page 1665; Chapter 30, "Portal Registry Classes," AbnPeopleCode, page 1665; Chapter 30, "Portal Registry Classes," DefaultChartNavigation, page 1669; Chapter 30, "Portal Registry Classes," TreeEffectiveDate, page 1673; Chapter 30, "Portal Registry Classes," TreeSetId, page 1675; Chapter 30, "Portal Registry Classes," TreeStructureName, page 1676 and Chapter 30, "Portal Registry Classes," TreeUserKeyValue, page 1677

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

TreeSetId

Description

Use this property to set or return the setID for the tree to be used as the SmartNavigation data source as a string.

Note. If the SmartNavigation data source is a rowset, this property is set to the Null string.

SmartNavigation passes the values of several tree-specific fields to the application via URL. Certain characters are inappropriate for use in a URL and must be avoided. When using a tree as a SmartNavigation data source, do not use any of the following characters in the tree name, setID, user key value, and tree branch values:

<table>
<tr><td>pound (#)</td><td>percent (%)</td><td>dollar ($)</td></tr>
<tr><td>ampersand (&)</td><td>plus (+)</td><td>comma (,)</td></tr>
<tr><td>forward slash/virgule (/)</td><td>colon (:)</td><td>semi-colon (;)</td></tr>
<tr><td>equals (=)</td><td>question mark (?)</td><td>at symbol (@)</td></tr>
<tr><td>space ( )</td><td>quotation marks(")</td><td>less than symbol (<)</td></tr>
<tr><td>greater than symbol (>)</td><td>left curly brace ({)</td><td>right curly brace (})</td></tr>
<tr><td>vertical bar/pipe (|)</td><td>backslash (\)</td><td>caret (^)</td></tr>
<tr><td>tilde (~)</td><td>left square bracket ([)</td><td>right square bracket (])</td></tr>
<tr><td>grave accent (`)</td><td></td><td></td></tr>
</table>

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," AbnContentProvider, page 1664; Chapter 30, "Portal Registry Classes," AbnDataSource, page 1665; Chapter 30, "Portal Registry Classes," AbnPeopleCode, page 1665; Chapter 30, "Portal Registry Classes," DefaultChartNavigation, page 1669; Chapter 30, "Portal Registry Classes," TreeEffectiveDate, page 1673; Chapter 30, "Portal Registry Classes," TreeName, page 1674; Chapter 30, "Portal Registry Classes," TreeStructureName, page 1676 and Chapter 30, "Portal Registry Classes," TreeUserKeyValue, page 1677

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

TreeStructureName

Description

Use this property to set or return the branch for the tree to be used as the SmartNavigation data source as a string.

Note. If the SmartNavigation data source is a rowset, this property is set to the Null string.

SmartNavigation passes the values of several tree-specific fields to the application via URL. Certain characters are inappropriate for use in a URL and must be avoided. When using a tree as a SmartNavigation data source, do not use any of the following characters in the tree name, setID, user key value, and tree branch values:

pound (#)
percent (%)
dollar ($)
ampersand (&)
plus (+)
comma (,)
forward slash/virgule (/)
colon (:)
semi-colon (;)
equals (=)
question mark (?)
at symbol (@)
space ( )
quotation marks(")
less than symbol (<)
greater than symbol (>)
left curly brace ({)
right curly brace (})
vertical bar/pipe (|)
backslash (\)
caret (^)
tilde (~)
left square bracket ([)
right square bracket (])
grave accent (`)

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," AbnContentProvider, page 1664; Chapter 30, "Portal Registry Classes," AbnDataSource, page 1665; Chapter 30, "Portal Registry Classes," AbnPeopleCode, page 1665; Chapter 30, "Portal Registry Classes," DefaultChartNavigation, page 1669; Chapter 30, "Portal Registry Classes," TreeEffectiveDate, page 1673; Chapter 30, "Portal Registry Classes," TreeName, page 1674; Chapter 30, "Portal Registry Classes," TreeSetId, page 1675 and Chapter 30, "Portal Registry Classes," TreeUserKeyValue, page 1677

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

TreeUserKeyValue

Description

Use this property to set or return the user key value (also known as the set control value) for the tree to be used as the SmartNavigation data source as a string.

Note. If the SmartNavigation data source is a rowset, this property is set to the Null string.

SmartNavigation passes the values of several tree-specific fields to the application via URL. Certain characters are inappropriate for use in a URL and must be avoided. When using a tree as a SmartNavigation data source, do not use any of the following characters in the tree name, setID, user key value, and tree branch values:

- pound (#)
- percent (%)
- dollar ($)
- ampersand (&)
- plus (+)
- comma (,)
- forward slash/virgule (/)
- colon (:)
- semi-colon (;)
- equals (=)
- question mark (?)
- at symbol (@)
- space ( )
- quotation marks(")
- less than symbol (<)
- greater than symbol (>)
- left curly brace ({)
- right curly brace (})
- vertical bar/pipe (|)
- backslash (\)
- caret (^)
- tilde (~)
- left square bracket ([)
- right square bracket (])
- grave accent (`)

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," AbnContentProvider, page 1664; Chapter 30, "Portal Registry Classes," AbnDataSource, page 1665; Chapter 30, "Portal Registry Classes," AbnPeopleCode, page 1665; Chapter 30, "Portal Registry Classes," DefaultChartNavigation, page 1669; Chapter 30, "Portal Registry Classes," TreeEffectiveDate, page 1673; Chapter 30, "Portal Registry Classes," TreeName, page 1674; Chapter 30, "Portal Registry Classes," TreeSetId, page 1675 and Chapter 30, "Portal Registry Classes," TreeStructureName, page 1676

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

ValidFrom

Description

This property returns or sets the date this folder is valid from as a string.

This property is read-write.

ValidTo

Description

This property returns or sets the date this folder is valid until as a string.

This property is read-write.

Note. The PortalRegistry API never uses the ValidTo and ValidFrom fields to determine what to return in a collection. You must check for these values in your application.

Folder Collection

The Folder Collection provides access to a collection of folders in a Folder object.

The Folder Collection is instantiated from the Folders Folder Class property.

See Chapter 30, "Portal Registry Classes," Folders, page 1669.

Folder Collection Methods

In this section, we discuss the Folder collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem*(FolderName)*

Description

The DeleteItem method deletes the folder object identified by *FolderName* from the database. If the folder contains other folders, all child folders and their contents are also deleted.

Warning! If you delete a folder, you delete *all* content in the folder. If you delete a folder that contains other folders, that is, a parent folder, all the child folders, and all the content references are deleted.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	*Description*
FolderName	Specify the name of a folder existing in the folder collection.

Returns

A Boolean value: True if the folder was deleted, False otherwise.

Example

```
If Not &MyFolderColl.DeleteItem("MYFOLDER") Then
   /* Folder not deleted. Do error checking */
End-If;
```

First

Syntax

First()

Description

The First method returns the first Folder object in the folder collection.

Parameters

None.

Returns

Folder object.

Example

```
&MyFolder = &MyCollection.First();
```

InsertItem

Syntax

InsertItem*(FolderName,Label)*

Description

The InsertItem method inserts the folder object identified by *FolderName* from the Folder Collection. You must specify both a name and a label for all folders. This method returns a reference to the new folder object. You must specify a unique *FolderName*, or you receive an error.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database *as soon as* the method is executed.

Parameters

Parameter	*Description*
FolderName	Specify the name of a folder existing in the folder collection. This parameter takes a string value.
Label	Specify a label for the new folder. This parameter takes a string value.

Returns

A reference to the new Folder object if the method executed successfully, null otherwise.

Example

```
&DeptHPFldr = &MyPortal.Folders.InsertItem("PORT0145", "HR Folder for Department⇒
 0145");
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the Folder object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing folder within the folder collection. If you specify an invalid name, the object is NULL. You must specify a name, not a label.

Returns

A folder object if successful, NULL otherwise.

Example

```
&DeptFldr = &MyPortal.RootFolder.Folders.ItemByName(&Dept_Name);
```

Next

Syntax

Next()

Description

The Next method returns the next Folder object in the Folder collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A folder object.

Example

```
Local ApiObject &MySession, &Root, &Folders, &MyFolder;

&MySession = %Session;
&MyPortal = &MySession.GetPortalRegistry("ADMIN");

&Root = &MyPortal.GetRoot();
&Folders = &Root.Folders;

&MyFolder = &Folders.First();

For &I = 1 to &Folders.Count
   /* Do processing on folders */
   If &I <> &Folders.Count
      &MyFolder = &Folders.Next();
   End-If;
End-For;
```

Folder Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Folder objects in the Folder Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Content Reference Class

The content reference class provides access to some kind of content. The type of content depends on the content reference.

The content reference objects are instantiated from other classes:

- From a PortalRegistry object with the FindCRefByURL, FindCRefForURL, or FindCRefByName properties.
- From a Content Reference Collection (instantiated from a folder) with the First, ItemByName or Next methods

See Chapter 30, "Portal Registry Classes," FindCRefByURL, page 1624; Chapter 30, "Portal Registry Classes," FindCRefByName, page 1623; Chapter 30, "Portal Registry Classes," FindCRefForURL, page 1625 and Chapter 30, "Portal Registry Classes," Content Reference Collection, page 1700.

See Chapter 30, "Portal Registry Classes," Adding a Content Reference, page 1814.

Content Reference Class Methods

In this section, we discuss the Content Reference class methods. The methods are discussed in alphabetical order.

CreateLink

Syntax

CreateLink*(LinkName,Label)*

Description

Use the CreateLink method to create a link quickly to the same content reference executing the method. The link by defaults assumes the parent folder is the same as the content reference's parent.

A CReflink object is returned if there is no error.

After you create a link you must use the Save method to save it to the database.

Parameters

Parameter	***Description***
LinkName	Specify the name of the link as a string. This parameter takes 30 characters. This is considered as ID of the link. It cannot start with number, and can not have special characters and spaces.
Label	Specify the label of the link as a string. This parameter takes 30 characters.

Returns

A reference to a ContentReference link object.

Example

```
&CRef = &Portal.FindCRefByName("MyCRef");
&Link = &CRef.CreateLink("Link Object");
&Link.Save();
```

See Also

Chapter 30, "Portal Registry Classes," Link Class, page 1736 and Chapter 30, "Portal Registry Classes," Save, page 1722

Save

Syntax

Save()

Description

The Save method saves any changes you made to the content reference, for example, a changed description. It also performs some validation.

Using this method also saves any changes you made to PermissionValue or AttributeValue objects associated with this content reference.

Parameters

None.

Returns

A Boolean value: True if the content reference and its associated objects saved successfully, False otherwise.

Example

```
If NOT(&MyCRef.Save()) Then
   /* save failed, do error processing */
End-If;
```

See Also

Chapter 30, "Portal Registry Classes," Save, page 1638 PortalRegistry method

Chapter 30, "Portal Registry Classes," Save, page 1663 Folder method

Content Reference Class Properties

In this section, we discuss the Content Reference class properties. The properties are discussed in alphabetical order.

AbsoluteContentURL

Description

This property returns the absolute content URL, that is, the content from the content servlet (psc).

This property is read-only.

Example

The following is an example absolute content URL:

```
http://serverx/psp/PS84/EMPLOYEEPORTAL/CRM/c/SFA.CUSTOMERINFO.GBL?page=CUST_⇒
DATA1&&Action=U&emplid=00001
```

AbsolutePortalURL

Description

This property returns the absolute content reference portal URL.

This property is read-only.

Example

```
http://serverx/psp/PS84/EMPLOYEEPORTAL/CRM/c/SFA.CUSTOMERINFO.GBL?page=CUST_⇒
DATA1&&Action=U&emplid=00001
```

AssignedPagelets

Description

This property returns an AssignedPagelet collection.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," AssignedPagelet Collection, page 1750

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this content reference object.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Attribute Collection, page 1707

Author

Description

This property returns the author (PeopleSoft user ID) for this content reference object as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the content reference has access to the content reference. This property takes a Boolean value. The default value for this property is True.

This property is read-write.

Authorized

Description

This property specifies whether the user is authorized to view this content reference.

This property is used when you access a particular content reference using FindCRefByURL, FindCRefForURL, or FindCRefByName. If you specified a valid content reference with either of these methods, a content reference is always returned, whether you are authorized to view it or not. This is the only property you can view from an object for which you are not authorized.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this content reference.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," FindCRefByURL, page 1624; Chapter 30, "Portal Registry Classes," FindCRefForURL, page 1625 and Chapter 30, "Portal Registry Classes," FindCRefByName, page 1623

CascadedPermissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the Permissions property instead. To access the role-based permissions, use the CascadedRolePermissions property.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1688; Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712 and Chapter 30, "Portal Registry Classes," Permissions, page 1692

CascadedRolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the RolePermissions property instead. To determine non role-based permissions, use the CascadedPermissions property.

Note. You *cannot* add any RolePermissionValue objects to a collection returned by the CascadedRolePermissions property. You can add values only to the collection returned by the RolePermissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1688; Chapter 30, "Portal Registry Classes," RolePermissions, page 1694 and Chapter 30, "Portal Registry Classes," RolePermissionValue Collection, page 1717

ContentProvider

Description

This property returns or sets the name of the node for the content reference as a string.

This property takes the following values:

Value	*Description*
Node name	Specify the exact name of the node.
LOCAL_NODE	Specify the node for the content reference as the node defined as the local node.

This property is read-write.

CreationDate

Description

This property returns the creation date for this content reference object as a string.

This property is read-only.

Data

Description

This property returns the data for this content reference. This property is valid only when the StorageType property is LOCL.

The length of this property depends on your system database limit for LONG fields.

This property is read-write.

Example

```
&MyData = &MyCRef.Data;
```

See Also

Chapter 30, "Portal Registry Classes," StorageType, page 1695

Description

Description

This property returns or sets the description for this content reference object as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

HtmlText

Description

This property returns the HTML text associated with this content reference as a string.

This property is read-only.

IsMobile

Description

Note. PeopleSoft Mobile Agent is a deprecated product. This mobile property currently exists for backward compatibility only.

This property returns True if this content reference is used with mobile applications, False otherwise.

This property is read-write.

IsVisible

Description

This property returns True if the Hide from Portal Navigation check box is not selected when the content reference is created. If the content reference is hidden from portal navigation, this property returns False.

Considerations Using IsVisible

If you do not specifically set this property on a new or copied content reference object, the IsVisible property is set as follows:

- If the ValidFrom date is less then or equal to that day's date, the IsVisible property is set to True.
- If there is no ValidTo date the IsVisible property is set to True.
- If the ValidTo date is greater then or equal to that day's date, the IsVisible property is set to True.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," ValidFrom, page 1699 and Chapter 30, "Portal Registry Classes," ValidTo, page 1700

Label

Description

This property returns or sets the label for this content reference object as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

Links

Description

This property returns a reference to a Link collection. This collection contains all the links that are associated with this content reference.

This property is read-only. However, a link collection is updated in realtime when a new link is created on the object.

See Also

Chapter 30, "Portal Registry Classes," Link Collection, page 1734

Name

Description

This property returns the name for this content reference object as a string. The name is a unique identifier for each content reference object.

Every content reference name must be unique in the portal, not just in the parent folder.

This property is read-only.

OwnerId

Description

This property returns or sets the owner ID of the content reference object as a string.

This property is read-write.

ParentName

Description

This property returns the parent folder name for this content reference object as a string.

This property is read-only.

Example

The following example uses the ParentName to return a folder object for the content reference.

```
&Folder = &Portal.FindFolderByName(&CRef.ParentName);
```

Path

Description

The Path property returns a path to this content reference, with each element of the path separated by a period. The path has the following syntax:

```
ContentReferenceLabel{Name}.[ChildContentReferenceLabel{Name}.]. . .
```

This property is read-only.

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for this content reference. To determine the permissions for all the parent objects (up to the root folder) use the CascadedPermissions property.

If you want to find role-based permissions for the content reference, use the RolePermission property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712; Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1687 and Chapter 30, "Portal Registry Classes," RolePermissions, page 1694

Product

Description

This property returns or sets the PeopleSoft product for this content reference object as a string.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property indicates whether a content reference is generally accessible, that is, if it will always be included in the general content reference collection. This property takes a Boolean value.

The default value for this property is False.

This property is read-write.

QualifiedURL

Description

Note. This property is being kept for backward compatibility only. If your code uses this property, the value returned is actually from the AbsoluteContentURL property. New applications should use the AbsoluteContentURL property instead.

See Also

Chapter 30, "Portal Registry Classes," AbsoluteContentURL, page 1685

RelativeURL

Description

This property returns the relative URL in the following format:

../../../Portal/Node/Content_Type/ContentID

For example, from the following URL:


```
http://mlee2038/servlets/psp/PS84/e_procurement/fdm/c/E_PRO.CheckOut.GBL?page=⇒
view&Setid=110&Custid=99
```

The RelativeURL returns the following:

```
e_procurement/fdm/c/E_PRO.CheckOut.GBL?page=view&Setid=110&Custid=99
```

This property is read-only.

RolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for this content reference. To determine the permissions for all the parent objects (up to the root folder) use the CascadedRolePermissions property.

If you want to find non role-based permissions for the content reference, use the Permission property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1688; Chapter 30, "Portal Registry Classes," Permissions, page 1692 and Chapter 30, "Portal Registry Classes," RolePermissionValue Collection, page 1717

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned content references is based on the sequence number. Use this property to reorder content references. This property takes a number value.

If there are duplicates in the sequence number, the content references are returned alphabetically.

The length of this property is 4 characters.

This property is read-write.

StorageType

Description

In general, content references contain information about where to get the content, and do not store the content. However, content references that are template or portal component types can have their content accessible directly from the content reference. In these cases, the Data property is valid and can be read or written, and the data is stored locally in the portal database.

StorageType	***Meaning***
LOCL	Local: Data property on content reference is valid.
RMTE	Remote: Data property is not valid.

When UsageType is a target this property must be set to RMTE and the URLType property should specify what format the Node and URL are in.

When UsageType is either a template or a portal component this property can be set to either LOCL or RMTE.

Note that when StorageType is LOCL, it specifies a static template or portal component. But, when StorageType is RMTE it can specify either:

- a dynamically generated template or portal component
- a static template or portal component

In both cases, the Node and URL properties specifies how to get the template or portal component.

Only templates and homepage tab content references can have StorageType LOCL. The StorageType is always LOCL for homepage tabs. For templates LOCL means that corresponding URL is stored in database, not retrieved by URL.

The following table indicates the usage type, and what type of storage is available.

UsageType	***RMTE***	***LOCL***
FRMT (Frame template)	X	X
HTMT (HTML template)	X	X
HPGT (Homepage tab)		X
HPGC (Pagelet)	X	

UsageType	*RMTE*	*LOCL*
TARG (Target)	X	

RMTE is the default value for a new content reference.

The length of this property is 4 characters.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," Data, page 1689

Template

Description

This property returns or sets the name of the template used with this content reference as a string. You must specify the name of an existing template.

This property uses the name *not* the label of a content reference.

This property is used only when the UsageType property is specified as Target (TARG) and the TemplateType property is specified as HTML.

To return a reference to the content reference that contains the template specified by this property, use the TemplateObject property.

The length of this property is 30 characters.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," UsageType, page 1699; Chapter 30, "Portal Registry Classes," TemplateType, page 1697 and Chapter 30, "Portal Registry Classes," TemplateObject, page 1696

TemplateObject

Description

This property returns a reference to a content reference object that contains the template specified by the Template property as a content reference. If no template is specified with Template, this property returns NULL.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Template, page 1696

TemplateType

Description

This property indicates whether a template should be used to wrap the content. This property takes a string value.

Values are:

Value	***Description***
HTML	An HTML template should wrap the data (URL) in the content reference.
NONE	No template should be used. The text should not be wrapped.

HTML is the default value on a new content reference.

Use the NONE value if the URL should not appear in the portal. An example is when the content reference is a template itself.

If the homepage template for the user is un-retrievable, the system tries to use the default user's template first, before resorting to the portal default template.

When the content reference describes content, this property should be set to HTML.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," Using Content References, page 1604

URL

Description

This property returns or sets the URL for this content reference object as a string. The URL returns *exactly*as it appears in the database.

If you're setting a URL, you must use a unique URL.

The absolute URL, that is, the URL from the node concatenated with this URL must be unique.

You receive an error if the URL your use is already registered.

To retrieve a qualified URL, that is, one that contains the node URI, use the AbsoluteContentURL property.

The length of this property depends on your system database limit for LONG fields.

This property is read-write.

The format of the value for this property depends on the setting of other properties.

See Also

Chapter 30, "Portal Registry Classes," AbsoluteContentURL, page 1685 and Chapter 30, "Portal Registry Classes," Using Content References, page 1604

URLType

Description

This property indicates what kind of URL is used to retreive the content. This property takes a string value.

Note. This property is used only for content that has the StorageType property set as Remote.

Values are:

URLType Value	***Meaning***
UEXT	URL points to a non-PeopleSoft (external) URL
UMPG	URL points to a PeopleSoft mobile page
UPGE	URL points to a component.
UPHP	URL points to a homepage tab
UPTM	URL points to a template
USCR	URL points to an iScript
UGEN	URL points to a generic PeopleSoft URL
WRKL	URL points to a Worklist URL

UPGE is the default value on a new content reference.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," StorageType, page 1695

UsageType

Description

This property indicates what the content reference is used for. Several other properties depend on this property.

This property takes a string value.

The length of this property is 4 characters.

Value	*Description*
FRMT	Frame template: the content reference is a frame-based template.
HTMT	HTML template: the content reference is an HTML template.
HPGT	Homepage tab: the content reference is a homepage tab.
HPGC	Pagelet: the content reference is a pagelet used in the homepage.
TARG	Target: the content reference is the target content its template determines what else must be loaded and how it will look.
LINK	Link: the content reference is a link.

TARG is the default value on a new content reference.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," Using Content References, page 1604

ValidFrom

Description

This property returns or sets the date this content reference is valid from as a string.

This property is read-write.

ValidTo

Description

This property returns or sets the date this content reference is valid until as a string.

This property is read-write.

Note. The PortalRegistry API never uses the ValidTo and ValidFrom fields to determine what to return in a collection. You must check for these values in your application.

Content Reference Collection

The Content Reference Collection provides access to the collection of content references in a Folder object.

The Content Reference Collection is instantiated from the ContentRefs Folder property.

See Chapter 30, "Portal Registry Classes," ContentRefs, page 1668.

Content Reference Collection Methods

In this section, we discuss the Content Reference collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

`DeleteItem`*`(ContentReferenceName)`*

Description

The DeleteItem method deletes the content reference object identified by *ContentReferenceName* from the content reference Collection.

If you delete a template for a content reference, and none of the other content references on a page have a template, the default template specified with the ContentProvider is used. If there's no template for the ContentProvider, the template for the PortalRegistry is used. However, if you delete the template for the content reference, the ContentProvider, and the PortalRegistry, you receive a runtime error.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	***Description***
ContentReferenceName	Specify the name of a content reference existing in the content reference collection.

Returns

A Boolean value: True if the content reference was deleted, False otherwise.

Example

```
If Not &MyCRef.DeleteItem("Test_CRef") Then
   /* can't delete test data. Do error processing */
End-if;
```

First

Syntax

First()

Description

The First method returns the first content reference object in the content reference collection.

Parameters

None.

Returns

A content reference object.

Example

```
&MyCRef = &MyCollection.First();
```


InsertItem

Syntax

InsertItem(*ContentReferenceName,ContentReferenceLabel, Node*, *URL)*

Description

The InsertItem method inserts the content reference object identified by *ContentReferenceName* into the content reference Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database as soon as the method is executed.

Parameters

Parameter	***Description***
ContentReferenceName	Specify the name of a new content reference. This parameter takes a string value. If you specify a name that already exists in the collection, you get an error.
ContentReferenceLabel	Specify a description of the new content reference. This is the translated value. This parameter takes a string value.
Node	Specify a node. This parameter takes a string value. If you specify a fully qualified value for *URL*, you can specify a NULL (that is, two quotation marks with no space between them ("")).
URL	Specify a URL that contains the content. The format of this parameter depends on other properties, such as the type of content reference, where the data is stored, and so on.

Returns

A reference to the new content reference object if the method executed successfully, NULL otherwise.

Example

The following example inserts an external content reference that is a template:

```
&URL = "t/" | &ITEMNAME;
&MyCRef = &CRefColl.InsertItem(&ITEMNAME, &ITEMLABEL, "", &URL);
```

The following example inserts a content reference where URLType is iScript (USCR):

```
&URL = "s/WEBLIB_PORTAL.FieldFormula.Portal_Trans_Dyn"
```



```
&MyCRef = &CRefColl.InsertItem(&ITEMNAME, &ITEMLABEL, "HRMS", &URL);
```

See Also

Chapter 30, "Portal Registry Classes," Using Content References, page 1604

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the content reference object with the name *Name*.

Parameters

Parameter	*Description*
Name	Specify the name of an existing content reference within the content reference collection. If you specify an invalid name, the object is NULL.

Returns

A content reference object if successful, NULL otherwise.

Example

```
&MyCRef = &CRefColl.ItemByName("PORTAL_ADMIN);
```

Next

Syntax

Next()

Description

The Next method returns the next content reference object in the content reference collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

content reference object.

Example

```
&MyCRef = &MyCollection.Next();
```

Content Reference Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of content reference objects in the content reference Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

AttributeValue Class

The AttributeValue class provides access to attributes associated with either folders or content references.

AttributeValue objects are instantiated from an Attribute Collection Methods with the First, InsertItem, ItemByName, or Next methods.

See Chapter 30, "Portal Registry Classes," Attribute Collection Methods, page 1707.

See Chapter 30, "Portal Registry Classes," Using Attributes, page 1819.

AttributeValue Class Properties

In this section, we discuss the AttributeValue class properties. The properties are discussed in alphabetical order.

Label

Description

This property returns the label of the AttributeValue as a string. This property works with the Translatable property. If Translatable is set to True, the value of Label can be translated.

The length of this property is 30 characters.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," Translatable, page 1706

Name

Description

This property returns the name of the AttributeValue as a string.

The length of this property is 30 characters.

This property is read-only.

Example

```
&AttrColl = &Folder.Attributes;
&Attr = &AttrColl.First();

   &Scroll = GetLevel0().GetRow(1).GetRowset(Scroll.PORTAL_FLDR_ATR);

   &I = 1;
   While All(&Attr)
      &Record = &Scroll.GetRow(&I).GetRecord(Record.PORTAL_FLDR_ATR);
      &Record.PORTAL_ATTR_NAM.Value = &Attr.Name;
      &Record.PORTAL_ATTR_VAL.Value = &Attr.Value;
      &Attr = &AttrColl.Next();
      /* need this check so we don't insert extra blank row */
      If All(&Attr) Then
         &Scroll.InsertRow(&I);
         &I = &I + 1;
      End-If;
End-While;
```

Translatable

Description

This property specifies if the AttributeValue is translatable. This property takes a Boolean value: True if the AttributeValue can be translated, False otherwise.

If this property is set to True, the value of the Label property can be translated.

Note. Regardless of the order in which attributes were entered, they are ordered according to their translatable property, that is, attributes that have this property set as True come first, followed by attributes that have this property set as False.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," Label, page 1705

Value

Description

This property returns the value of the AttributeValue as a string.

The length of this property depends on your system database limit for LONG fields.

This property is read-write.

Example

To specify more than a single value for an AttributeValue, you can specify several values separated by a semicolon. For example:

```
&MyAtt.value = "401k;benefits;dependants;HR";
```

Attribute Collection

The Attribute Collection provides access to the collection of Attribute in a Folder or a content reference object.

An Attribute Collection is instantiated from other classes as follows:

- From a content reference object with the Attributes property.
- From a Folder object with the Attributes property.
- From a PageletCategory object with the Attributes property.
- From a Pagelet object with the Attributes property.

See Chapter 30, "Portal Registry Classes," Attributes, page 1686.

See Chapter 30, "Portal Registry Classes," Attributes, page 1666.

See Chapter 30, "Portal Registry Classes," Attributes, page 1768.

See Chapter 30, "Portal Registry Classes," Attributes, page 1778.

Attribute Collection Methods

In this section, we discuss the Attribute collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

`DeleteItem`*`(AttributeValueName)`*

Description

The DeleteItem method deletes the AttributeValue object identified by *AttributeValueName* from the Attribute Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	*Description*
AttributeValueName	Specify the name of an AttributeValue existing in the Attribute Collection.

Returns

A Boolean value: True if the AttributeValue was deleted, False otherwise.

First

Syntax

```
First()
```

Description

The First method returns the first AttributeValue object in the Attribute Collection.

Parameters

None.

Returns

An AttributeValue object.

Example

```
&MyAttributeValue = &MyCollection.First();
```

InsertItem

Syntax

```
InsertItem(AttributeValueName)
```

Description

The InsertItem method inserts the AttributeValue object identified by *AttributeValueName* from the Attribute Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	*Description*
AttributeValueName	Specify the name of an AttributeValue existing in the Attribute Collection.

Returns

A reference to the new AttributeValue object if the method executed successfully, NULL otherwise.

Example

```
For &I = 1 To &Rowset.ActiveRowCount
    If &CompName = "PORTAL_CREF_ADM" Then
       &Record = &Rowset.GetRow(&I).GetRecord(Record.PORTAL_CATR_DV);
    Else
       &Record = &Rowset.GetRow(&I).GetRecord(Record.PORTAL_FATR_DV);
    End-If;

    If &Record.PORTAL_ATTR_NAM.Value <> "" Then
       &Attr = &AttrColl.InsertItem(&Record.PORTAL_ATTR_NAM.Value);
       &Attr.Value = &Record.PORTAL_ATTR_VAL.Value;
    End-If;
End-For;
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the AttributeValue object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing AttributeValue within the Attribute Collection. If you specify an invalid name, the object is NULL.

Returns

An AttributeValue object if successful, NULL otherwise.

Example

```
&Attr = &AttrColl.ItembyName("RELLINK");
```

Next

Syntax

```
Next()
```

Description

The Next method returns the next AttributeValue object in the Attribute Collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

AttributeValue object.

Example

```
&MyAttributeValue = &MyCollection.Next();
```

Attribute Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of AttributeValue objects in the Attribute Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

PermissionValue Class

The PermissionValue provides access to permission lists associated with folders, content references, PageletCategory objects, or Pagelets.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions or CascadedRolePermissions property. You can add values only to the collection returned by the Permissions or RolePermissions property.

PermissionValue objects are instantiated from the following:

- A PermissionValue Collection with the First, InsertItem, ItemByName, or Next methods.
- A RolePermissionValue Collection with the First, InsertItem, ItemByName, or Next methods.

See Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712.

See Chapter 30, "Portal Registry Classes," Using Security, page 1591.

See Chapter 30, "Portal Registry Classes," Setting Permissions Using the PermissionValue Object, page 1817.

PermissionValue Class Properties

In this section we discuss the Cascade and Name properties.

Cascade

Description

This property indicates whether the current permission should be granted to all child folders.

This property is valid only with folders, not with content references.

This property takes a Boolean value. The default value for a new PermissionValue object is False.

This property is read-write.

Name

Description

Specify the name of a permission list as the name of this object. You must specify a permission list that has already been created. This property takes a string value.

The length of this property is 30 characters.

This property is read-write.

PermType

Description

Specify the type of the permission. Values are:

Value	*Description*
P	Specify a non role-based permission.
R	Specify a role-based permission.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," RolePermissionValue Collection, page 1717 and Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712

PermissionValue Collection

A PermissionValue collection is returned by the following:

- CascadedPermissions and Permissions folder property
- CascadedPermissions and Permissions content reference property
- CascadedPermissions and Permissions PageletCategory object property

- CascadedPermissions and Permissions Pagelet property

What is contained in the PermissionValue collection depends on the property that created it.

See Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1667 and Chapter 30, "Portal Registry Classes," Permissions, page 1672.

See Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1687 and Chapter 30, "Portal Registry Classes," Permissions, page 1692.

See Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1769 and Chapter 30, "Portal Registry Classes," Permissions, page 1772.

See Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1779 and Chapter 30, "Portal Registry Classes," Permissions, page 1782.

See Chapter 30, "Portal Registry Classes," Using Security, page 1591.

PermissionValue Collection Methods

In this section, we discuss the PermissionValue collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem*(PermissionValueName)*

Description

The DeleteItem method deletes the PermissionValue object identified by *PermissionValueName* from the PermissionValue Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	***Description***
PermissionValueName	Specify the name of a PermissionValue existing in the PermissionValue collection.

Returns

A Boolean value: True if the PermissionValue was deleted, False otherwise.

Example

```
If Not &MyPValColl.DeleteItem("ALLPNLS") Then
   /* do error processing */
End-If;
```

First

Syntax

First()

Description

The First method returns the first PermissionValue object in the PermissionValue collection.

Parameters

None.

Returns

A PermissionValue object.

Example

```
&MyPermissionValue = &MyCollection.First();
```

InsertItem

Syntax

InsertItem*(PermissionValueName)*

Description

The InsertItem method inserts the PermissionValue object identified by *PermissionValueName* into the PermissionValue Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can only add values to the collection returned by the Permissions property.

Parameters

Parameter	*Description*
PermissionValueName	Specify the name of an existing permission list.

Returns

A reference to the new PermissionValue object if the method executed successfully, NULL otherwise.

Example

```
&MyPermV = &MyPermVColl.InsertItem("ALLPNLS");

If Not &MyPermV Then
   /* do error processing */
End-If;
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the PermissionValue object with the name *Name*.

Parameters

Parameter	*Description*
Name	Specify the name of an existing PermissionValue within the PermissionValue collection. If you specify an invalid name, the object is NULL.

Returns

A PermissionValue object if successful, NULL otherwise.

Example

```
&MyPVal = &MyPValColl.ItemByName("CUSTOMER");
```

Next

Syntax

```
Next()
```

Description

The Next method returns the next PermissionValue object in the PermissionValue collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

PermissionValue object.

Example

```
&MyPermissionValue = &MyCollection.Next();
```

PermissionValue Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of PermissionValue objects in the PermissionValue Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```


RolePermissionValue Collection

A RolePermissionValue collection is returned by the following:

- CascadedRolePermissions and RolePermissions folder property

  See Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1668.

  See Chapter 30, "Portal Registry Classes," RolePermissions, page 1673.

- CascadedRolePermissions and RolePermissions content reference property

  See Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1688.

  See Chapter 30, "Portal Registry Classes," RolePermissions, page 1694.

- CascadedRolePermissions and RolePermissions content reference link property

  See Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1726.

  See Chapter 30, "Portal Registry Classes," RolePermissions, page 1731.

What is contained in the RolePermissionValue collection depends on the property that created it.

RolePermissionValue Collection Methods

In this section, we discuss the RolePermissionValue collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

`DeleteItem`*`(RolePermissionValueName)`*

Description

The DeleteItem method deletes the RolePermissionValue object identified by *RolePermissionValueName* from the RolePermissionValue Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	*Description*
RolePermissionValueName	Specify the name of a RolePermissionValue existing in the RolePermissionValue collection.

Returns

A Boolean value: True if the RolePermissionValue was deleted, False otherwise.

Example

```
If Not &MyPValColl.DeleteItem("ALLPNLS") Then
   /* do error processing */
End-If;
```

First

Syntax

First()

Description

The First method returns the first RolePermissionValue object in the RolePermissionValue collection.

Parameters

None.

Returns

A RolePermissionValue object.

Example

```
&MyRolePermissionValue = &MyCollection.First();
```


InsertItem

Syntax

InsertItem*(RolePermissionValueName)*

Description

The InsertItem method inserts the RolePermissionValue object identified by *RolePermissionValueName* into the RolePermissionValue Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Note. You *cannot* add any RolePermissionValue objects to a collection returned by the CascadedRolePermissions property. You can only add values to the collection returned by the RolePermissions property.

Parameters

Parameter	***Description***
RolePermissionValueName	Specify the name of an existing role permission list.

Returns

A reference to the new RolePermissionValue object if the method executed successfully, NULL otherwise.

Example

```
&MyPermV = &MyPermVColl.InsertItem("ALLPNLS");

If Not &MyPermV Then
   /* do error processing */
End-If;
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the RolePermissionValue object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing RolePermissionValue within the RolePermissionValue collection. If you specify an invalid name, the object is NULL.

Returns

A RolePermissionValue object if successful, NULL otherwise.

Example

```
&MyPVal = &MyPValColl.ItemByName("CUSTOMER");
```

Next

Syntax

Next()

Description

The Next method returns the next RolePermissionValue object in the RolePermissionValue collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

RolePermissionValue object.

Example

```
&MyRolePermissionValue = &MyCollection.Next();
```


RolePermissionValue Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of RolePermissionValue objects in the RolePermissionValue Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

ContentReference Links

A content reference link is a ContentReference object that delegates most of its properties to another content reference. It has its own separate properties, as well as a reference to the linked cref.

A content reference link is instantiated from the following:

- CreateLink content reference method
- CreateContentRefLink PortalRegistry method

Note. The CreateLink method creates a new link. To instantiate an existing link use the FindLinkByName method.

See Also

Chapter 30, "Portal Registry Classes," CreateContentRefLink, page 1619

Chapter 30, "Portal Registry Classes," FindCRefLinkByName, page 1627

Chapter 30, "Portal Registry Classes," CreateLink, page 1683

ContentReference Links Methods

In this section, we discuss the ContentReference link methods. The methods are discussed in alphabetical order.

Delete

Syntax

Delete()

Description

Use the delete method to delete the link from the database.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The link is not marked for deletion, then actually deleted later. The link is deleted from the database as soon as the method is executed.

Parameters

None.

Returns

A Boolean value: True if the content reference link and its associated objects are deleted successfully, False otherwise.

Save

Syntax

Save()

Description

The Save method saves any changes you made to the content reference link, for example, a changed description or ValidFrom date.

After you create a link using either the CreateLink or CreateContentRefLink methods you must use the Save method to save the link to the database.

Using this method also saves any changes you made to PermissionValue or AttributeValue objects associated with this content reference link.

Parameters

None.

Returns

A Boolean value: True if the content reference link and its associated objects saved successfully, False otherwise. If False is returned, a message is also written to the PSMessage collection.

Example

```
If NOT(&MyCRefLink.Save()) Then
   /* save failed, do error processing */
End-If;
```

ContentReference Links Properties

In this section, we discuss the ContentReference link properties. The properties are discussed in alphabetical order.

AbsoluteContentURL

Description

This property returns the absolute content URL, that is, the content from the content servlet (psc). The content is displayed without any portal template.

This property is read-only.

Example

The following is an example absolute content URL:

```
http://serverx/psc/PS84/EMPLOYEEPORTAL/CRM/c/SFA.CUSTOMERINFO.GBL?page=CUST_⇒
DATA1&&Action=U&emplid=00001
```


AbsolutePortalURL

Description

This property returns the absolute content reference link portal URL. This URL also contains the content in the portal template.

This property is read-only.

Example

```
http://serverx/psp/PS84/EMPLOYEEPORTAL/CRM/c/SFA.CUSTOMERINFO.GBL?page=CUST_⇒
DATA1&&Action=U&emplid=00001
```

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this content reference link object.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," Attribute Collection, page 1707

Author

Description

This property returns the author (PeopleSoft user ID) for this content reference link as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the content reference link has access to the content reference link. This property takes a Boolean value. The default value for this property is true.

This property is read-write.

Authorized

Description

This property specifies whether the user is authorized to view this content reference link.

This property is used when you access a particular content reference link using FindCRefLinkByURL. If you specified a valid content reference link, a content reference link is always returned, whether you are authorized to view it or not. This is the only property you can view from an object that you are not authorized tom and the content reference is empty.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this content reference link.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," FindCRefLinkByName, page 1627

CascadedPermissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the Permissions property instead.

Note. You cannot add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712 and Chapter 30, "Portal Registry Classes," Permissions, page 1692

CascadedRolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the RolePermissions property instead. To determine the non role-based permissions of the object use the Permissions property instead.

Note. You cannot add any PermissionValue objects to a collection returned by the CascadedRolePermissions property. You can add values only to the collection returned by the RolePermissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1726; Chapter 30, "Portal Registry Classes," RolePermissions, page 1731 and Chapter 30, "Portal Registry Classes," RolePermissionValue Collection, page 1717

ContentProvider

Description

This property returns the name of the node for the content reference link as a string.

This property is read-only.

CreationDate

Description

This property returns the creation date for this content reference link object as a string.

This property is read-only.

Data

Description

This property returns the data for this content reference link. This property is valid only when the StorageType property is LOCL.

The length of this property depends on your system database limit for LONG fields.

This property is read-write.

Example

```
&MyData = &MyCReflink.Data;
```

See Also

Chapter 30, "Portal Registry Classes," StorageType, page 1695

Description

Description

This property returns or sets the description for this content reference link object as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

IsMobile

Description

Note. PeopleSoft Mobile Agent is a deprecated product. This mobile property currently exists for backward compatibility only.

This property returns True if this content reference link is used with mobile applications, False otherwise.

This property is read-write.

IsVisible

Description

This property returns True if the Hide from Portal Navigation check box is not selected when the content reference link is created. If the content reference link is hidden from portal navigation, this property returns False. Also, if the date associated with the link is not valid (is not within the valid to and valid from dates,) this property returns False.

Considerations Using IsVisible

If you do not specifically set this property on a new or copied content reference link, the IsVisible property is set as follows:

- If the ValidFrom date is less then or equal to that day's date, the IsVisible property is set to True.
- If there is no ValidTo date the IsVisible property is set to True.
- If the ValidTo date is greater then or equal to that day's date, the IsVisible property is set to True.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," ValidFrom, page 1699 and Chapter 30, "Portal Registry Classes," ValidTo, page 1700

Label

Description

This property returns or sets the label for this content reference link object as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

Name

Description

This property returns the name for this content reference link object as a string. The name is a unique identifier for each content reference link object. The name cannot start with a number, and it cannot have spaces and special characters.

Every content reference link name must be unique in the portal, not just in the parent folder.

This property is read-write.

OwnerId

Description

This property returns or sets the owner ID of the content reference link object as a string.

This property is read-write.

ParentName

Description

This property returns the parent folder name for this content reference link object as a string.

You can use this property to move the content reference link to another folder.

This property is read-write.

Example

The following example uses the ParentName to return a folder object for the content reference link.

```
&Folder = &Portal.FindFolderByName(&CReflink.ParentName);
```

Path

Description

The Path property returns a path to this content reference link, with each element of the path separated by a period. The path has the following syntax:

```
Root{PORTAL_ROOT_OBJECT}.LinkFolder {<Id of links folder>}.CrefLink {<Id of the⇒
 cref link> }
```

This property is read-only.

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for this content reference link. To access the role-based permissions, use the RolePermissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712; Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1687 and Chapter 30, "Portal Registry Classes," RolePermissions, page 1731

Product

Description

This property returns or sets the PeopleSoft product for this content reference link object as a string.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property specifies whether the link is public or private. This property takes a Boolean value: true if the content reference link is public and there are no permission lists associated with it. If it is false, the link uses permission list control.

For links, this values is same as that of the linked content reference. If the content reference is public, the link is public and can be viewed by anyone logged on to the system.

This property is read-only.

RelativeURL

Description

This property returns the relative URL in the following format:

`../../../`*`Portal/Node/Content_Type/ContentID`*

For example, from the following URL:

```
http://mlee2038/servlets/psp/PS84/e_procurement/fdm/c/E_PRO.CheckOut.GBL?page=⇒
view&Setid=110&Custid=99
```

The RelativeURL returns the following:

```
e_procurement/fdm/c/E_PRO.CheckOut.GBL?page=view&Setid=110&Custid=99
```

This property is read-only.

RolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for this content reference link. To access the non role-based permissions, use the Permissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," CascadedRolePermissions, page 1726; Chapter 30, "Portal Registry Classes," Permissions, page 1729 and Chapter 30, "Portal Registry Classes," RolePermissionValue Collection, page 1717

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned content reference links is based on the sequence number. Use this property to reorder content reference links. This property takes a number value.

If there are duplicates in the sequence number, the content reference links are returned alphabetically.

The length of this property is 4 characters.

This property is read-write.

Template

Description

This property returns or sets the name of the template used with this content reference link as a string. You must specify the name of an existing template.

This property uses the name, not the label, of a content reference link.

This property is used only when the UsageType property is specified as Target (TARG) and the TemplateType property is specified as HTML.

To return a reference to the content reference link that contains the template specified by this property, use the TemplateObject property.

The length of this property is 30 characters.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," UsageType, page 1699; Chapter 30, "Portal Registry Classes," TemplateType, page 1697 and Chapter 30, "Portal Registry Classes," TemplateObject, page 1696

TemplateObject

Description

This property returns a reference to a content reference link object that contains the template specified by the Template property as a content reference link. If no template is specified with Template, this property returns a null value.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Template, page 1696

TemplateType

Description

This property indicates whether a template should be used to wrap the content. This property takes a string value.

Values are:

Value	***Description***
HTML	An HTML template should wrap the data (URL) in the content reference.
NONE	No template should be used. The text should not be wrapped.

HTML is the default value on a new content reference.

Use the NONE value if the URL should not appear in the portal. An example is when the content reference link is a template itself.

If the homepage template for the user is un-retrievable, the system tries to use the default user's template first, before resorting to the portal default template.

When the content reference link describes content, this property should be set to HTML.

This property is read-write.

See Also

Chapter 30, "Portal Registry Classes," Using Content References, page 1604

URL

Description

This property returns the URL for this content reference link object as a string. The content reference link derives the URL from the content reference it is linked to. The URL of the link is same as that of the linked content reference.

A query string value is added on the content reference URL to make the link unique. If the URL of the linked content reference changes, this URL is changed.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," AbsoluteContentURL, page 1685 and Chapter 30, "Portal Registry Classes," Using Content References, page 1604

URLType

Description

This property returns what kind of URL is used to retreive the content.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," StorageType, page 1695

UsageType

Description

This property returns what the content reference link is used for.

The value for this property is same as that of the content reference when the link is vieved as a content reference.

The usage type of the link is LINK in the database.

A content reference link can be created only for content references that have a type of TARG.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Using Content References, page 1604

ValidFrom

Description

This property returns or sets the date this content reference link is valid from as a string.

This property is read-write.

ValidTo

Description

This property returns or sets the date this content reference link is valid until as a string.

This property is read-write.

Note. The PortalRegistry API never uses the ValidTo and ValidFrom fields to determine what to return in a collection. You must check for these values in your application.

Link Collection

A Links collection is returned by the Links content reference property.

See Also

Chapter 30, "Portal Registry Classes," Links, page 1691

Link Collection Methods

In this section, we discuss the link collection methods. The methods are discussed in alphabetical order.

First

Syntax

`First()`

Description

The First method returns the first link object in the link collection.

Parameters

None.

Returns

A link object.

Next

Syntax

`Next()`

Description

The Next method returns the next link object in the collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A link object.

Link Collection Property

In this section, we discuss the link collection property Count.

Count

Description

This property returns the number of link objects in the link collection, as a number.

This property is read-only.

Link Class

A Link object is returned by the First and Next Link collection methods.

See Also

Chapter 30, "Portal Registry Classes," Link Collection Methods, page 1734

Link Properties

In this section, we discuss the link properties. The properties are discussed in alphabetical order.

LinksObjectName

Description

This property returns the object name of the link as a string.

This property is read-only.

LinksObjectType

Description

This property returns the object type as a string. Valid value is CONTENTREF.

This property is read-only.

LinksPortalName

Description

This property returns the name of the portal from which the link originated.

This property is read-only.

TabDefinition Class

The TabDefinition class provides access to the homepage tab defined for the portal.

TabDefinition objects are instantiated from a TabDefinition Collection with the First, InsertItem, ItemByName, or Next methods.

See Chapter 30, "Portal Registry Classes," TabDefinition Collection, page 1745.

TabDefinition Class Methods

In this section, we discuss the Save method.

Save

Syntax

Save()

Description

The Save method saves any changes you made to the TabDefinition, for example, a changed description. It also performs some validation.

Using this method also saves any changes you made to DynamicCategories and AssignedPagelets objects associated with this TabDefinition.

Parameters

None.

Returns

A Boolean value: True if the TabDefinition and its associated object saved successfully, False otherwise.

Example

```
If NOT(&MyTabDefn.Save()) Then
   /* save failed, do error processing */
End-If;
```

See Also

Chapter 30, "Portal Registry Classes," Save, page 1638 PortalRegistry method

Chapter 30, "Portal Registry Classes," Save, page 1663 Folder method

Chapter 30, "Portal Registry Classes," Save, page 1684 ContentReference method

TabDefinition Class Properties

In this section, we discuss the TabDefinition class properties. The properties are discussed in alphabetical order.

AssignedPagelets

Description

This property returns a reference to an AssignedPagelet Collection for this TabDefinition object.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," AssignedPagelet Collection, page 1750

AvailableCategories

Description

This property returns a reference to an AvailableCategory Collection for this TabDefinition object.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," AvailableCategory Collection, page 1756

AvailablePagelets

Description

This property returns a reference to an AvailablePagelet Collection for this TabDefinition object.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," AvailablePagelet Collection, page 1760

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this TabDefinition object.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Attribute Collection, page 1707

Author

Description

This property returns the author (PeopleSoft user ID) for this TabDefinition object as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the TabDefinition has access to the TabDefinition. This property takes a Boolean value. The default value for this property is True.

This property is read-write.

ColumnLayout

Description

This property indicates whether the tab layout is for two or three columns. This property takes a string value.

The values for this property are:

Value	***Description***
2	Two-column table
3	Three-column table

This property is read-write.

CreationDate

Description

This property returns the creation date for this TabDefinition object as a string.

This property is read-only.

Description

Description

This property returns or sets the description for this TabDefinition object as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

DynamicCategories

Description

This property returns a reference to a DynamicCategory Collection for this TabDefinition object.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," DynamicCategory Collection, page 1763

HelpID

Description

This property returns the help system identifier for this tab. It enables the help system to provide tab-specific help. This property takes a string value.

This property is read-write.

HtmlText

Description

This property returns the HTML text associated with this tab definition as a string.

This property is read-only.

IsHideActionBar

Description

This property hides all pagelet action bar images for all pagelets used on this tab. This property overrides any properties for the pagelets used on this tab. This property takes a Boolean value: True, hide images, False, display images

This property is read-write.

IsLayoutLocked

Description

This property indicates whether the user can change the tab's default number of columns. This property takes a Boolean value: True, the user can change the tab columns, False, the columns can't be changed by the user.

This property is read-write.

IsRenameable

Description

This property indicates whether the tab's label can be changed by the user. This property takes a Boolean value: True, the user can change the label, False, the user can't.

This property is read-write.

Label

Description

This property returns or sets the label for this TabDefinition object as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

LayoutBehavior

Description

This property indicates whether the user can change when and how the tab displays. This property takes a string value.

The values are:

Value	***Description***
4OPT	The user can choose to add this tab to their homepage. It doesn't show up by default.
3DEF	The user sees this tab the first time they log in, however, they can remove this tab if they'd like.
2REQ	The user can't remove this tab from their homepage. However, they are allowed to change its sequence or order on the page.
1FIX	The user can't remove this tab or change its position on the page.

This property is read-write.

Name

Description

This property returns the name for this TabDefinition object as a string. The name is a unique identifier for each TabDefinition object.

Every TabDefinition name must be unique in the portal, not just in the parent folder.

This property is read-only.

OwnerId

Description

This property returns or sets the owner ID of the TabDefinition object as a string.

This property is read-write.

Product

Description

This property returns or sets the PeopleSoft product for this TabDefinition object as a string.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property indicates whether a TabDefinition is generally accessible, that is, if it is always included in the general TabDefinition collection. This property takes a Boolean value.

The default value for this property is False.

This property is read-write.

QualifiedURL

Description

This property returns an absolute URL. If the TabDefinition has a ContentProvider associated with it, the URI from the ContentProvider is concatenated with the URL from the TabDefinition. If there is no ContentProvider, this property returns the text in the URL property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," URL, page 1697

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned TabDefinitions is based on the sequence number. Use this property to reorder TabDefinitions. This property takes a number value.

If there are duplicates in the sequence number, the TabDefinitions are returned alphabetically.

The length of this property is 4 characters.

This property is read-write.

StyleSheet

Description

This property defines the style sheet to use for this tab. This property takes a string value. This property is 30 characters long.

If no style sheet is specified, the default style sheet PSSTYLEDEF is used.

This property is read-write.

ValidFrom

Description

This property returns or sets the date this TabDefinition is valid from as a string.

This property is read-write.

ValidTo

Description

This property returns or sets the date this TabDefinition is valid until as a string.

This property is read-write.

Note. The PortalRegistry API never uses the ValidTo and ValidFrom fields to determine what to return in a collection. You must check for these values in your application.

TabDefinition Collection

A TabDefinition collection is the collection of all TabDefinitions for a portal.

A TabDefinition collection is returned by the TabDefinitions PortalRegistry property.

See Chapter 30, "Portal Registry Classes," TabDefinitions, page 1642.

TabDefinition Collection Methods

This section describes the TabDefinition collection methods in alphabetical order.

DeleteItem

Syntax

`DeleteItem``(`*`TabDefinitionName`*`)`

Description

The DeleteItem method deletes the TabDefinition object identified by *TabDefinitionName* from the TabDefinition Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	*Description*
TabDefinitionName	Specify the name of a TabDefinition existing in the TabDefinition collection.

Returns

A Boolean value: True if the TabDefinition was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("TabDefnTest") Then
   /* do error processing */
End-If;
```

First

Syntax

```
First()
```

Description

The First method returns the first TabDefinition object in the TabDefinition collection.

Parameters

None.

Returns

A TabDefinition object.

Example

```
&MyTabDefinition = &MyCollection.First();
```


InsertItem

Syntax

InsertItem*(TabDefinitionName)*

Description

The InsertItem method inserts the TabDefinition object identified by *TabDefinitionName* into the TabDefinition Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database *as soon as* the method is executed.

Parameters

Parameter	***Description***
TabDefinitionName	Specify the name of an existing permission list.

Returns

A reference to the new TabDefinition object if the method executed successfully, NULL otherwise.

Example

```
&MyTabDef = &MyColl.InsertItem("Empl_Homepage");

If Not &MyTabDef Then
   /* do error processing */
End-If;
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the TabDefinition object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing TabDefinition within the TabDefinition collection. If you specify an invalid name, the object is NULL.

Returns

A TabDefinition object if successful, NULL otherwise.

Example

```
&MyTebDefn = &MyColl.ItemByName("Empl_Homepage");
```

Next

Syntax

Next()

Description

The Next method returns the next TabDefinition object in the TabDefinition collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

TabDefinition object.

Example

```
&MyTabDefinition = &MyCollection.Next();
```

TabDefinition Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of TabDefinition objects in the TabDefinition Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

AssignedPagelet Class

An AssignedPagelet is a pagelet object that has been assigned to a particular tab. This pagelet also contains the layout data that is specific to the pagelet and tab.

AssignedPagelet objects are instantiated from an AssignedPagelet Collection with the First, InsertItem, ItemByName, or Next methods.

See Chapter 30, "Portal Registry Classes," AssignedPagelet Collection, page 1750.

AssignedPagelet Class Properties

In this section, we discuss the AssignedPagelet class properties. The properties are discussed in alphabetical order.

Column

Description

This property returns the column in which this pagelet is displayed, as a number.

This property is read-write.

LayoutBehavior

Description

This property indicates whether the user can change when and how the pagelet displays. This property takes a string value.

The values are:

Value	*Description*
4OPT	The user can choose to add this pagelet to their tab. It doesn't show up by default. The user can remove and reposition this pagelet on the tab.
3DEF	The user sees this pagelet the first time they log in, however, they can remove or reposition this pagelet if they'd like.
2REQ	The user can't remove this pagelet from the tab. However, they are allowed to reposition the pagelet on the tab.
1FIX	The user can't remove this pagelet or change its position on the tab.

This property is read-write.

PageletName

Description

This property returns the name for this Pagelet object as a string. The name is a unique identifier for each Pagelet object.

This property is read-only.

Row

Description

This property returns the row in which this pagelet is displayed, as a number.

This property is read-write.

AssignedPagelet Collection

An AssignedPagelet collection is returned by the AssignedPagelet TabDefinition method.

It contains a collection of all the pagelets explicitly assigned to the tab.

See Chapter 30, "Portal Registry Classes," AssignedPagelets, page 1738.

See Chapter 30, "Portal Registry Classes," AssignedPagelet Class, page 1749.

AssignedPagelet Collection Methods

In this section, we discuss the AssignedPagelet collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

`DeleteItem`*`(PageletName)`*

Description

The DeleteItem method deletes the AssignedPagelet object identified by *PageletName* from the AssignedPagelet Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	***Description***
PageletName	Specify the name of an AssignedPagelet existing in the AssignedPagelet collection.

Returns

A Boolean value: True if the AssignedPagelet was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Weather_411") Then
   /* do error processing */
End-If;
```

First

Syntax

`First``()`

Description

The First method returns the first AssignedPagelet object in the AssignedPagelet collection.

Parameters

None.

Returns

An AssignedPagelet object.

Example

```
&MyAssignedPagelet = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*PageletName*,*Column*,*Row*, *LayoutBehavior*)

Description

The InsertItem method inserts the AssignedPagelet object identified by the parameters into the AssignedPagelet Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	***Description***
PageletName	Specify the name of the pagelet that you want to insert.
Column	Specify the column that you want this pagelet to display in, as a number.
Row	Specify the row that you want this pagelet to display in, as a number. You can specify a zero if you don't know which row.
LayoutBehavior	Specify whether the user can change when and how the pagelet displays. This property takes a string value. The values are:

Value	Description
4OPT	The user can choose to add this pagelet to their tab. It doesn't show up by default. The user can remove and reposition this pagelet on the tab.
3DEF	The user sees this pagelet the first time they log in, however, they can remove or reposition this pagelet if they'd like.
2REQ	The user can't remove this pagelet from the tab. However, they are allowed to reposition the pagelet on the tab.
1FIX	The user can't remove this pagelet or change its position on the tab.

Returns

A reference to the new AssignedPagelet object if the method executed successfully, NULL otherwise.

Example

```
&MyPagelet = &MyColl.InsertItem("Weather_411");

If Not &MyPagelet Then
   /* do error processing */
End-If;
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the AssignedPagelet object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing AssignedPagelet within the AssignedPagelet collection. If you specify an invalid name, the object is NULL.

Returns

An AssignedPagelet object if successful, NULL otherwise.

Example

```
&MyPagelet = &MyColl.ItemByName("Dictionary");
```

Next

Syntax

Next()

Description

The Next method returns the next AssignedPagelet object in the AssignedPagelet collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

An AssignedPagelet object.

Example

```
&MyAssignedPagelet = &MyCollection.Next();
```

AssignedPagelet Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of AssignedPagelet objects in the AssignedPagelet Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

AvailableCategory Class

An AvailableCategory class is composed of a category name and an AvailablePagelet Collection, that is, a collection of all the available pagelets associated with that category.

AvailableCategory objects are instantiated from an AvailableCategory Collection with the First, ItemByName, or Next methods.

See Chapter 30, "Portal Registry Classes," AvailableCategory Collection, page 1756.

AvailableCategory Class Properties

In this section, we discuss the AvailableCategory class properties. The properties are discussed in alphabetical order.

AvailablePagelets

Description

This property returns a reference to an AvailablePagelet Collection associated with the specified CategoryName.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," AvailablePagelet Collection, page 1760

CategoryName

Description

Use this property to return the name of a category to be associated with the TabDefinition, as a string.

This property is read-only.

AvailableCategory Collection

An AvailableCategory is composed of a category name and an AvailablePagelet Collection, that is, a collection of all the available pagelets associated with that category.

An AvailableCategory collection is returned by the AvailableCategories TabDefinition property.

See Chapter 30, "Portal Registry Classes," AvailableCategory Collection, page 1756.

AvailableCategory Collection Methods

In this section, we discuss the AvailableCategory collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first AvailableCategory object in the AvailableCategory collection.

Parameters

None.

Returns

An AvailableCategory object.

Example

```
&MyAvailableCategory = &MyCollection.First();
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the AvailableCategory object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing AvailableCategory within the AvailableCategory collection. If you specify an invalid name, the object is NULL.

Returns

An AvailableCategory object if successful, NULL otherwise.

Example

```
&MyCat = &MyColl.ItemByName("Dictionary");
```

Next

Syntax

Next()

Description

The Next method returns the next AvailableCategory object in the AvailableCategory collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

AvailableCategory object.

Example

```
&MyAvailableCategory = &MyCollection.Next();
```

AvailableCategory Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of AvailableCategory objects in the AvailableCategory Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

AvailablePagelet Class

An AvailablePagelet is a pagelet object can be assigned to a particular tab. This includes all pagelets from the dynamic categories, and assigned pagelets. Each pagelet also contains the layout data that is specific to that pagelet.

AvailablePagelet objects are instantiated from an AvailablePagelet Collection with the First, ItemByName, or Next methods.

See Chapter 30, "Portal Registry Classes," AvailablePagelet Collection, page 1760.

AvailablePagelet Class Properties

In this section, we discuss the AvailablePagelet class properties. The properties are discussed in alphabetical order.

CategoryLabel

Description

This property turns the label of the category to which the pagelet is assigned, as a string.

This property is read-only.

CategoryName

Description

This property returns the name of the category to which the pagelet is assigned, as a string.

This property is read-only.

Column

Description

This property returns the column in which this pagelet is displayed, as a number.

This property is read-only.

LayoutBehavior

Description

This property indicates whether the user can change when and how the pagelet displays. This property takes a string value.

The values are:

Value	***Description***
4OPT	The user can choose to add this pagelet to their tab. It doesn't show up by default. The user can remove and reposition this pagelet on the tab.
3DEF	The user sees this pagelet the first time they log in, however, they can remove or reposition this pagelet if they'd like.
2REQ	The user can't remove this pagelet from the tab. However, they are allowed to reposition the pagelet on the tab.

Value	***Description***
1FIX	The user can't remove this pagelet or change its position on the tab.

This property is read only.

PageletLabel

Description

This property returns the label for this property as a string.

This property is read-only.

PageletName

Description

This property returns the name for this Pagelet object as a string. The name is a unique identifier for each Pagelet object.

This property is read-only.

Row

Description

This property returns the row in which this pagelet is displayed, as a number.

This property is read-only.

AvailablePagelet Collection

An AvailablePagelet collection is returned by the AvailablePagelet TabDefinition method.

It contains a collection of all the pagelets that could be assigned to the tab. It's an aggregation of the pagelets in the AssignedPagelets collection and all the pagelets in each of the categories in the DynamicCategories collection. Pagelet category sequence numbers, followed by pagelet sequence numbers, then name, sort this collection.

If you have two pagelets assigned with the same name (that is, an assigned pagelet and a dynamic category pagelet), the assigned pagelet takes precedence over the dynamic category one, and is the only one listed.

See Chapter 30, "Portal Registry Classes," AvailablePagelets, page 1739.

See Chapter 30, "Portal Registry Classes," AvailablePagelet Class, page 1758.

AvailablePagelet Collection Methods

In this section, we discuss the AvailablePagelet collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first AvailablePagelet object in the AvailablePagelet collection.

Parameters

None.

Returns

An AvailablePagelet object.

Example

```
&MyAvailablePagelet = &MyCollection.First();
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the AvailablePagelet object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing AvailablePagelet within the AvailablePagelet collection. If you specify an invalid name, the object is NULL.

Returns

An AvailablePagelet object if successful, NULL otherwise.

Example

```
&MyPagelet = &MyColl.ItemByName("DICTIONARY");
```

Next

Syntax

Next()

Description

The Next method returns the next AvailablePagelet object in the AvailablePagelet collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

AvailablePagelet object.

Example

```
&MyAvailablePagelet = &MyCollection.Next();
```

AvailablePagelet Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of AvailablePagelet objects in the AvailablePagelet Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

DynamicCategory Collection

A DynamicCategory collection contains a collection of PageletCategory names associated with a TabDefinition. The collection is initially ordered by category name.

When a PageletCategory is dynamic, it is immediately available to an end-user.

A DynamicCategory Collection is instantiated by the DynamicCategories property of a TabDefinition.

See Chapter 30, "Portal Registry Classes," DynamicCategories, page 1740.

DynamicCategory Collection Methods

In this section, we discuss the DynamicCategory collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem*(Name)*

Description

The DeleteItem method deletes the PageletCategory object identified by *Name* from the DynamicCategory Collection. This does *not* delete the PageletCategory from the database, just from the DynamicCategory collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Note. If you delete a DynamicCategory, tabs with the attribute PORTAL_HPTAB_DYN lists must be searched for and updated (if necessary) to reflect the deleted category.

Parameters

Parameter	***Description***
Name	Specify the name of an existing category in the DynamicCategory collection.

Returns

A Boolean value: True if the PageletCategory was deleted from the DynamicCategory collection, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Dictionaries") Then
   /* do error processing */
End-If;
```

First

Syntax

First()

Description

The First method returns the name of the first PageletCategory in the DynamicCategory collection.

Parameters

None.

Returns

A string.

Example

```
&MyDynamicCategory = &MyCollection.First();
```


InsertItem

Syntax

InsertItem*(Name)*

Description

The InsertItem method inserts the PageletCategory object identified by *Name* into the DynamicCategory Collection. This does *not* insert the PageletCategory into the database, just into the DynamicCategory collection. After a PageletCategory is marked as dynamic, it is immediately available to the end-user.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	*Description*
Name	Specify the name of the DynamicCategory to insert.

Returns

A string containing the name of the new DynamicCategory object if the method executed successfully, Null otherwise.

Example

```
&MyCat = &MyColl.InsertItem("User_Info");
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the name of the PageletCategory object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing DynamicCategory within the DynamicCategory collection. If you specify an invalid name, this method returns a Null string.

Returns

A string.

Example

```
&MyPagelet = &MyColl.ItemByName("Dictionary");
```

Next

Syntax

Next()

Description

The Next method returns the name of the next PageletCategory object in the DynamicCategory collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A string.

Example

```
&MyDynamicCategory = &MyCollection.Next();
```


DynamicCategory Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of PageletCategory names in the DynamicCategory Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

PageletCategory Class

A PageletCategory is a group of related pagelets, such as Weather, News, Reference, and so on. Each PageletCategory has a collection of pagelets associated with it.

A PageletCategory is instantiated from a PageletCategory Collection with the First, InsertItem, ItemByName, or Next methods.

All PageletCategory objects that are dynamic are contained in the DynamicCategory collection. The methods from this collection return the names of the PageletCategory objects only, not references to the objects themselves.

See Chapter 30, "Portal Registry Classes," PageletCategory Collection, page 1773.

PageletCategory Class Method

In this section, we discuss the Save method.

Save

Syntax

```
Save()
```


Description

The Save method saves any changes you made to the PageletCategory, for example, a changed description or sequence number.

Parameters

None.

Returns

A Boolean value: True if the PageletCategory object is successfully saved, False otherwise.

Example

```
If Not(&MyPageletCategory.Save()) Then
   /* do error checking */
End-If;
```

See Also

Chapter 30, "Portal Registry Classes," Save, page 1638 PortalRegistry method

Chapter 30, "Portal Registry Classes," Save, page 1684 Folder method

PageletCategory Class Properties

In this section, we discuss the PageletCategory class properties. The properties are discussed in alphabetical order.

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this PageletCategory object.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Attribute Collection, page 1707

Author

Description

This property returns the author (PeopleSoft user ID) for this PageletCategory as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the PageletCategory has access to the PageletCategory. This property takes a Boolean value. The default value for this property for a newly created object is True. This property is not cascaded.

This property is read-write.

Authorized

Description

This property specifies whether the user is authorized to view this PageletCategory.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this PageletCategory.

This property is read-only.

CascadedPermissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the permissions for all the child and parent objects (up to the root PageletCategory). To determine only the permissions of the object use the Permission property instead.

Note. The PORTAL_PAGELETS folder is the parent folder for all PageletCategories. Its security is set to public. PeopleSoft does not recommend cascading any permissions from this folder object. Cascade permissions only from the pagelet category (folder). You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712 and Chapter 30, "Portal Registry Classes," Permissions, page 1672

CreationDate

Description

This property returns the creation date for this PageletCategory as a string.

This property is read-only.

Description

Description

This property returns or sets the description for this PageletCategory as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

Label

Description

This property returns or sets the label for this PageletCategory as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

Name

Description

This property returns the name for this PageletCategory as a sting. The name is a unique identifier for each PageletCategory.

Every PageletCategory name must be unique in across the portal, not just in the parent PageletCategory.

This property is *not* translatable. However, the values for the Label and Desrciption properties are translatable.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Label, page 1770

OwnerId

Description

This property returns or sets the owner ID of the PageletCategory object as a string.

This property is read-write.

Pagelets

Description

This property returns a reference to a Pagelet Collection for this PageletCategory.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Pagelet Collection, page 1785

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the permissions for this PageletCategory.

Note. The PORTAL_PAGELETS folder is the parent folder for all PageletCategories. Its security is set to public. PeopleSoft does not recommend cascading any permissions from this folder object. Cascade permissions only from the pagelet category (folder).

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712

Product

Description

This property returns or sets the PeopleSoft product for this PageletCategory as a string.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property indicates whether a PageletCategory is generally accessible, that is, if this property is set to True, any user can access the PageletCategory. This property is not cascaded.

This property takes a Boolean value.

The default value for this property for a newly created object is False.

This property is read-write.

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned PageletCategory objects is based on the sequence number. Use this property to reorder PageletCategory objects.

If there are duplicates in the sequence number, the PageletCategory objects are returned in alphabetical order.

The length of this property is 4 characters.

This property is read-write.

PageletCategory Collection

The PageletCategory Collection provides access to a collection of PageletCategory objects in a PageletCategory object.

The PageletCategory Collection is instantiated from the PageletCategories PortalRegistry property.

See Chapter 30, "Portal Registry Classes," PageletCategories, page 1642.

PageletCategory Collection Methods

In this section, we discuss the PageletCategory collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

`DeleteItem`*`(PageletCategoryName)`*

Description

The DeleteItem method deletes the PageletCategory object identified by *PageletCategoryName* from the database.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	Description
PageletCategoryName	Specify the name of a PageletCategory existing in the PageletCategory collection.

Returns

A Boolean value: True if the PageletCategory was deleted, False otherwise.

Example

```
If Not &MyPageletCategoryColl.DeleteItem("MYPAGELETCATEGORY") Then
   /* PageletCategory not deleted. Do error checking */
End-If;
```

First

Syntax

```
First()
```

Description

The First method returns the first PageletCategory object in the PageletCategory collection.

Parameters

None.

Returns

PageletCategory object.

Example

```
&MyPageletCategory = &MyCollection.First();
```


InsertItem

Syntax

`InsertItem`*`(PageletCategoryName,Label)`*

Description

The InsertItem method inserts the PageletCategory object identified by *PageletCategoryName* from the PageletCategory Collection. You must specify both a name and a label for each PageletCategory. This method returns a reference to the new PageletCategory object. You must specify a unique *PageletCategoryName*, or you receive an error.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database *as soon as* the method is executed.

Parameters

Parameter	*Description*
PageletCategoryName	Specify the name of a PageletCategory existing in the PageletCategory collection. This parameter takes a string value.
Label	Specify a label for the new PageletCategory. This parameter takes a string value.

Returns

A reference to the new PageletCategory object if the method executed successfully, False otherwise.

ItemByName

Syntax

`ItemByName`*`(Name)`*

Description

The ItemByName method returns the PageletCategory object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing PageletCategory within the PageletCategory collection. If you specify an invalid name, the object is NULL. You must specify a name, not a label.

Returns

A PageletCategory object if successful, NULL otherwise.

Next

Syntax

```
Next()
```

Description

The Next method returns the next PageletCategory object in the PageletCategory collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A PageletCategory object.

PageletCategory Collection Property

In this section, we discuss the PageletCategory collection properties. The properties are discussed in alphabetical order.

Count

Description

This property returns the number of PageletCategory objects in the PageletCategory Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Pagelet Class

A pagelet represents a URL that renders a section of content for a homepage. Pagelets are assigned to specific categories.

A pagelet is instantiated from the First, InsertItem, ItemByName, and Next Pagelet Collection methods.

See Chapter 30, "Portal Registry Classes," Pagelet Collection, page 1785.

Pagelet Class Method

In this section, we discuss the Save method.

Save

Syntax

Save()

Description

The Save method saves any changes you made to the Pagelet, for example, a changed description or sequence number.

Parameters

None.

Returns

A Boolean value: True if the Pagelet object is successfully saved, False otherwise.

Example

```
If Not(&MyPagelet.Save()) Then
   /* do error checking */
End-If;
```

See Also

Chapter 30, "Portal Registry Classes," Save, page 1638 PortalRegistry method

Chapter 30, "Portal Registry Classes," Save, page 1684 ContentReference method

Chapter 30, "Portal Registry Classes," Save, page 1767 PageletCategory method

Pagelet Class Properties

In this section, we discuss the Pagelet class properties. The properties are discussed in alphabetical order.

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this pagelet object.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," Attribute Collection, page 1707

Author

Description

This property returns the author (PeopleSoft user ID) for this pagelet object as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the pagelet has access to the pagelet. This property takes a Boolean value. The default value for this property is True.

This property is read-write.

Authorized

Description

This property specifies whether the user is authorized to view this pagelet.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this pagelet.

This property is read-only.

CascadedPermissions

Description

This property returns a PermissionValue collection. This collection contains the value of the permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the Permission property instead.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712 and Chapter 30, "Portal Registry Classes," Permissions, page 1692

ContentProvider

Description

This property returns or sets the name of the node for the pagelet as a string.

If no node was specified when the pagelet was created, this property returns Null.

This property is read-write.

CreationDate

Description

This property returns the creation date for this pagelet object as a string.

This property is read-only.

DefaultColumn

Description

This property returns or sets the default column this pagelet displays in. This property takes a number value.

The default value for this property is 1.

This property is read-write.

Description

Description

This property returns or sets the description for this pagelet object as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

EditPageContentProvider

Description

This is the node part of the URL to the page that enables a user to personalize a page. The default value is an empty string.

This property is read-write.

EditPageQueryString

Description

This is the query string part of the URL to the page that enables a user to personalize a page. The default value for this property is an empty string.

This property is read-write.

HelpID

Description

This property returns the help system identifier for this pagelet. It enables the help system to provide pagelet specific help. This property takes a string value.

This property is read-write.

IsHideMinimize

Description

This property specifies if the minimize image icon displays. This property takes a Boolean value: True, display the icon, False, hide the icon. The default value is True.

This property is read-write.

Label

Description

This property returns or sets the label for this pagelet object as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

Name

Description

This property returns the name for this pagelet object as a string. The name is a unique identifier for each pagelet object.

Every pagelet name must be unique in the portal, not just in the parent folder.

This property is not translatable. However, the values for the Label and Description properties are.

This property is read-only.

OwnerId

Description

This property returns or sets the owner ID of the pagelet object as a string.

This property is read-write.

ParentName

Description

This property returns the parent folder name for this pagelet object as a string.

This property is read-write.

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the permissions for this pagelet. To determine the permissions for all the parent objects (up to the root folder) use the CascadedPermissions property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," PermissionValue Collection, page 1712 and Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1687

Product

Description

This property returns or sets the PeopleSoft product for this pagelet object as a string.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property indicates whether a pagelet is generally accessible, that is, if it will always be included in the general pagelet collection. This property takes a Boolean value.

The default value for this property is False.

This property is read-write.

QualifiedURL

Description

This property returns an absolute URL. If the pagelet has a node associated with it, the URI from the node is concatenated with the URL from the pagelet. If there is no ContentProvider, this property returns the text in the URL property.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," URL, page 1697

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned pagelets is based on the sequence number. Use this property to reorder pagelets. This property takes a number value.

If there are duplicates in the sequence number, the pagelets are returned alphabetically.

The length of this property is 4 characters.

This property is read-write.

URL

Description

This property returns or sets the URL for this pagelet object as a string. The URL returns *exactly* as it appears in the database.

If you're setting a URL, you must use a unique URL.

The absolute URL, that is, the URL from the ContentProvider concatenated with this URL must be unique.

You receive an error if the URL you use is already registered.

To retrieve a qualified URL, that is, one that contains the node URI, use the AbsoluteContentURL property.

The length of this property depends on your system database limit for LONG fields.

The format of the value for this property depends on the setting of other properties.

This property is read-write.

URLType

Description

This property indicates what kind of PeopleCode definition is used to retrieve the content of the pagelet. This property takes a string value.

Values are:

URLType Value	***Meaning***
UEXT	URL points to a non-PeopleSoft (external) URL
UPGE	URL points to a component.
UGEN	URL points to a generic PeopleSoft URL
USCR	URL points to an iScript

UPGE is the default value on a new pagelet.

This property is read-write.

Pagelet Collection

The Pagelet Collection provides access to the collection of pagelets in a category.

The Pagelet Collection is instantiated from the Pagelets PageletCategory property.

See Chapter 30, "Portal Registry Classes," Pagelets, page 1771.

Pagelet Collection Methods

In this section, we discuss the Pagelet collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem*(PageletName)*

Description

The DeleteItem method deletes the pagelet object identified by *PageletName* from the pagelet Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database as soon as the method is executed.

Parameters

Parameter	***Description***
PageletName	Specify the name of a pagelet existing in the pagelet collection.

Returns

A Boolean value: True if the pagelet was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Test_Pagelet") Then
   /* can't delete test data. Do error processing */
End-if;
```


First

Syntax

`First``()`

Description

The First method returns the first pagelet object in the pagelet collection.

Parameters

None.

Returns

A reference to a pagelet object if successful, NULL otherwise.

Example

```
&MyCRef = &MyCollection.First();
```

InsertItem

Syntax

`InsertItem`*`(PageletName,Label,NodeName, URL)`*

Description

The InsertItem method inserts the pagelet object identified by *PageletName* into the pagelet Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database *as soon as* the method is executed.

Parameters

Parameter	*Description*
PageletName	Specify the name of a new pagelet. This parameter takes a string value. If you specify a name that already exists in the collection, you receive an error.
Label	Specify the label for the new pagelet. This parameter takes a string value.
NodeName	Specify a node. This parameter takes a string value. If you specify a fully qualified URL, you can specify a Null (that is, two quotation marks with no space between them ("")).
URL	Specify a URL that contains the content. The format of this parameter depends on other properties, such as the type of pagelet, where the data is stored, and so on.

Returns

A reference to the new pagelet object if the method executed successfully, NULL otherwise.

See Also

Chapter 30, "Portal Registry Classes," Using Content References, page 1604

ItemByName

Syntax

`ItemByName`*`(Name)`*

Description

The ItemByName method returns the pagelet object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing pagelet within the pagelet collection. If you specify an invalid name, the object will be NULL.

Returns

A reference to a pagelet object if successful, NULL otherwise.

Example

```
&MyPagelet = &CRefColl.ItemByName("PORTAL_ADMIN);
```

Next

Syntax

Next()

Description

The Next method returns the next pagelet object in the pagelet collection. You can only use this method after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a pagelet object if successful, NULL otherwise.

Example

```
&MyPagelet = &MyCollection.Next();
```

Pagelet Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of pagelet objects in the pagelet Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

UserHomepage Class

The UserHomepage represents a homepage for a user. It contains all the tabs and pagelets the user has selected for their homepage.

A UserHomepage is instantiated from the Homepage PortalRegistry property.

See Chapter 30, "Portal Registry Classes," Homepage, page 1640.

UserHomepage Class Method

In this section, we discuss the Save method.

Save

Syntax

Save()

Description

The Save method saves any changes you made to the UserHomepage, for example, a changed greeting.

Using this method also saves any changes you've made to any UserTab or SelectedPagelet objects associated with this UserHomepage.

Parameters

None.

Returns

A Boolean value: True if the UserHomepage object is successfully saved, False otherwise.

Example

```
If Not(&MyHP.Save()) Then
   /* do error checking */
End-If;
```

See Also

Chapter 30, "Portal Registry Classes," Save, page 1638 PortalRegistry class method

Chapter 30, "Portal Registry Classes," Save, page 1663 Folder class method

UserHomepage Class Properties

In this section, we discuss the UserHomepage properties. The properties are discussed in alphabetical order.

Greeting

Description

This property is a user-definable message that appears on their homepage when they login, such as "My Homepage" or "Welcome." This property takes a text string. The length of this character is 254.

This property is read-write.

UserId

Description

This property returns the UserId for this UserHomepage as a string.

This property is read-only.

UserTab

Description

This property returns a reference to a UserTab collection.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," UserTab Collection, page 1793

UserTab Class

A UserTab is a personalized tab for a specific user. This user can potentially change the pagelets that appear on the tab, the name of the tab, and the order of the tab. All these changes can be allowed or denied on the TabDefinitions Class. The UserTab is used in place of the TabDefinition whenever a user has personalized a tab.

A UserTab object is instantiated from the First, ItemByName, InsertItem, or Next UserTab Collection methods.

See Chapter 30, "Portal Registry Classes," UserTab Collection, page 1793.

UserTab Class Properties

In this section, we discuss the UserTab class properties. The properties are discussed in alphabetical order.

ColumnLayout

Description

This property specifies how many columns the UserTab contains. This property takes a numeric value. The values are:

Value	*Description*
2	Two-column layout
3	Three-column layout. This is the default value.

This property is read-write.

Label

Description

This property returns or sets the label for this UserTab object as a string. This value is initially set to the Label of the TabDefinition.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

QualifiedURL

Description

This property returns an absolute URL as a string. This URL includes the URI from the TabDefinition's node and the URL that points to the actual content of the tab.

This property is read-only.

SelectedPagelets

Description

This property returns a SelectedPagelets Collection, containing the pagelets that have been selected by the user to appear on this tab.

This property is read-only.

See Also

Chapter 30, "Portal Registry Classes," SelectedPagelet Collection, page 1798

SequenceNumber

Description

This property specifies the order that this tab should appear as on the homepage. This property takes a number value. This value is initially set by the referenced TabDefinition.

This property is read-write.

TabName

Description

This property returns the tab name as a sting.

This property is read-only.

UserTab Collection

The UserTab collection is the collection of personalized tabs for a user.

The UserTab Collection is instantiated from the UserTabs UserHomepage property.

See Chapter 30, "Portal Registry Classes," UserTab, page 1790.

UserTab Collection Methods

In this section, we discuss the UserTab collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem*(UserTabName)*

Description

The DeleteItem method deletes the UserTab object identified by *UserTabName* from the UserTab Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	***Description***
UserTabName	Specify the name of a UserTab existing in the UserTab collection.

Returns

A Boolean value: True if the UserTab was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Test_UserTab") Then
   /* can't delete test data. Do error processing */
End-if;
```


First

Syntax

`First``()`

Description

The First method returns the first UserTab object in the UserTab collection.

Parameters

None.

Returns

A UserTab object.

Example

```
&MyUserTab = &MyCollection.First();
```

InsertItem

Syntax

`InsertItem`*`(UserTabName)`*

Description

The InsertItem method inserts the UserTab object identified by *UserTabName* into the UserTab Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

The new user tab must be unique within the portal.

Parameters

Parameter	***Description***
UserTabName	Specify the name of a new UserTab. This parameter takes a string value. If you specify a name that already exists in the collection, you get an error.

Returns

A reference to the new UserTab object if the method executed successfully, NULL otherwise.

ItemByName

Syntax

`ItemByName`*`(Name)`*

Description

The ItemByName method returns the UserTab object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing UserTab within the UserTab collection. If you specify an invalid name, the object is NULL.

Returns

A UserTab object if successful, NULL otherwise.

Example

```
&MyUserTab = &MyColl.ItemByName("PORTAL_ADMIN");
```

Next

Syntax

`Next``()`

Description

The Next method returns the next UserTab object in the UserTab collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A UserTab object.

Example

```
&MyUserTab = &MyCollection.Next();
```

UserTab Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of UserTab objects in the UserTab Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SelectedPagelet Class

The SelectedPagelet class contains data on pagelets selected by the user that display on this tab.

A SelectedPagelet object is instantiated by the First, InsertItem, ItemByName, and Next SelectedPagelet Collection methods.

See Chapter 30, "Portal Registry Classes," SelectedPagelet Collection, page 1798.

SelectedPagelet Class Properties

In this section, we discuss the SelectedPagelet class properties. The properties are discussed in alphabetical order.

CategoryName

Description

This property returns the category of the pagelet as a string.

This property is read-only.

Column

Description

This property returns the number of the column the pagelet displays in, as a number.

The values are:

Value	***Description***
1	First column
2	Second column
3	Third column

This property is read-write.

IsMinimized

Description

This property specifies whether the pagelet is displayed minimized. This property takes a Boolean value: True, the pagelet is minimized, False otherwise.

This property is read-write.

PageletName

Description

This property returns the name of the pagelet as a string.

This property is read-only.

Row

Description

This property specifies which row the pagelet is displayed in, as a number.

This property is read-write.

SelectedPagelet Collection

The SelectedPagelet collection contains a collection of all the selected pagelets for a tab.

A SelectedPagelet collection is instantiated by the SelectedPagelets UserTab property.

See Chapter 30, "Portal Registry Classes," SelectedPagelets, page 1792.

SelectedPagelet Collection Methods

In this section, we discuss the SelectedPagelet collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

`DeleteItem`*`(SelectedPageletName)`*

Description

The DeleteItem method deletes the SelectedPagelet object identified by *SelectedPageletName* from the SelectedPagelet Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	*Description*
SelectedPageletName	Specify the name of a SelectedPagelet existing in the SelectedPagelet collection.

Returns

A Boolean value: True if the SelectedPagelet was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Test_SelectedPagelet") Then
   /* can't delete test data. Do error processing */
End-if;
```

First

Syntax

First()

Description

The First method returns the first SelectedPagelet object in the SelectedPagelet collection.

Parameters

None.

Returns

A SelectedPagelet object.

Example

```
&MySelectedPagelet = &MyCollection.First();
```


InsertItem

Syntax

`InsertItem`_`(SelectedPageletName,Column,Row)`_

Description

The InsertItem method inserts the SelectedPagelet object identified by *SelectedPageletName* into the SelectedPagelet Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	***Description***
SelectedPageletName	Specify the name of a new SelectedPagelet. This parameter takes a string value. If you specify a name that already exists in the collection, you get an error.
Column	Specify the column number for this pagelet.
Row	Specify the row number for this pagelet.

Returns

A reference to the new SelectedPagelet object if the method executed successfully, NULL otherwise.

ItemByName

Syntax

`ItemByName`_`(Name)`_

Description

The ItemByName method returns the SelectedPagelet object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing SelectedPagelet within the SelectedPagelet collection. If you specify an invalid name, the object is NULL.

Returns

A SelectedPagelet object if successful, NULL otherwise.

Example

```
&MySelectedPagelet = &MyColl.ItemByName("PORTAL_ADMIN");
```

Next

Syntax

Next()

Description

The Next method returns the next SelectedPagelet object in the SelectedPagelet collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A SelectedPagelet object.

Example

```
&MySelectedPagelet = &MyCollection.Next();
```

SelectedPagelet Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of SelectedPagelet objects in the SelectedPagelet Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Favorite Class

A favorite represents a content reference the user wants to keep a shortcut to. It contains the name of the content reference and the label the user wants to see displayed for this favorite. Labels must be unique in each favorite collection.

Use the Save method of the Favorite class to save any changes you made to any favorite objects.

A favorite object is instantiated by the First, InsertItem, ItemByName, and Next Favorite Collection methods.

See Chapter 30, "Portal Registry Classes," Favorite Collection, page 1804.

Favorite Class Properties

In this section, we discuss the Favorite class properties. The properties are discussed in alphabetical order.

CRefName

Description

This property returns the name of the content reference as a string.

This property is read-only.

Label

Description

This property specifies the description of the favorite that's displayed to the user. This label must be unique for the favorite collection.

This property is read-write.

QualifiedURL

Description

This property returns the qualified URL for the content reference based on the Node and the URL for the name of the content reference, as a string.

This property is read-only.

SequenceNumber

Description

This property returns the sequence number used to order favorites.

This property is read-write.

URL

Description

This property returns the relative URL for this favorite as a string.

For example, the following code:

```
&URL = &MyFavorite.URL;
```

could return the following string:

```
c/PORTAL_PERS_HOMEPAGE.PORTAL_HOMEPAGE.GBL?PORTALPARAM_PAGE=CONTENT&tab=⇒
DEFAULT&PortalFavorite=QEDMO
```

This property is read-write.

Favorite Collection

The favorite collection provides access to the favorites for a particular user.

A favorite collection is instantiated by the Favorites PortalRegistry property.

See Chapter 30, "Portal Registry Classes," Favorites, page 1640.

Favorite Collection Methods

In this section, we discuss the Favorite collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

`DeleteItem``(`*`FavoriteLabel`*`)`

Description

The DeleteItem method deletes the Favorite object identified by *FavoriteLabel* from the Favorite Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	***Description***
FavoriteLabel	Specify the label of a Favorite existing in the Favorite collection, that is, the text used to identify the favorite to the end-user.

Returns

A Boolean value: True if the Favorite was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("My Local Test") Then
   /* can't delete test data. Do error processing */
End-if;
```

First

Syntax

First()

Description

The First method returns the first Favorite object in the Favorite collection.

Parameters

None.

Returns

A Favorite object.

Example

```
&MyFav = &MyCollection.First();
```

InsertItem

Syntax

InsertItem*(FavoriteLabel,FavoriteName)*

Description

The InsertItem method inserts the Favorite object identified by *FavoriteName* into the Favorite Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted to the database *as soon as* the method is executed.

Parameters

Parameter	*Description*
FavoriteLabel	Specify a label for the new Favorite. This is the text by which the favorite is identified to the end-user.
FavoriteName	Specify the name of a new Favorite. This parameter takes a string value. If you specify a name that already exists in the collection, you get an error.

Returns

A reference to the new Favorite object if the method executed successfully, NULL otherwise.

ItemByLabel

Syntax

ItemByLabel*(FavoriteLabel)*

Description

The ItemByLabel method returns the Favorite object with the label *FavoriteLabel.* The label is the text used to identify the favorite to the end-user.

Parameters

Parameter	*Description*
FavoriteLabel	Specify the label of an existing Favorite within the Favorite collection. If you specify an invalid favorite, the object returned is Null.

Returns

A Favorite object if successful, Null otherwise.

Example

```
&MyFavorite = &MyColl.ItemByLabel("My Local Login");
```


Next

Syntax

Next()

Description

The Next method returns the next Favorite object in the Favorite collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A Favorite object.

Example

```
&MyFavorite = &MyCollection.Next();
```

Save

Syntax

Save()

Description

Use the Save method to save any changes you made to the Favorite collection.

Parameters

None.

Returns

A Boolean value: True, the collection saved successfully, False otherwise.

Favorite Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Favorite objects in the Favorite Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count
```

PortalRegistry Classes Example

There are several actions you may want to perform using a PortalRegistry, such as adding a folder, changing permissions, and so on. The following examples discuss the most common tasks. In addition, there are example programs of accessing the PortalRegistry classes using language environments other than PeopleCode.

Changing PortalRegistry Properties

This example shows how to change the default template used by a PortalRegistry. The following is the complete code sample: the steps explain each line.


```
Local ApiObject &MySession;
Local ApiObject &MyPortal, &MyPortalColl;

/* Access the current session */

&MySession = %Session;
If NOT &MySession Then
   /* do error processing */
End-if;

&MyPortalColl = &MySession.FindPortalRegistries();

For &I = 1 to &MyPortalColl.Count

   &MyPortal = &MyPortalColl.Item(&I);

   If &MyPortal.DefaultTemplate = "HR99" Then
      &MyPortal.DefaultTemplate = "HR00"
      &MyPortal.Save();
   End-If

End-For;
```

To change PortalRegistry properties:

1. Get a Session object.

   Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.)

   ```
   &MySession = %Session;
   ```

2. Get a PortalRegistry collection.

   Use the FindPortalRegistries method with no parameters to return a collection of all the PortalRegistries because you want to check all the registries for the invalid template.

   ```
   &MyPortalColl = &MySession.FindPortalRegistries();
   ```

   Use the Count property on the collection to loop through all the registries.

   ```
   For &I = 1 to &MyPortalColl.Count
   ```

3. Get a PortalRegistry object.

   You can access a PortalRegistry object from the PortalRegistry collection using the Item method.

   ```
   &MyPortal = &MyPortalColl.Item(&I);
   ```

4. Check for the invalid template and correct if necessary.

   Use the DefaultTemplate property both to check for the invalid template name, and to set the correct template. Save the PortalRegistry when you make a correction.

   ```
   If &MyPortal.DefaultTemplate = "HR99" Then
      &MyPortal.DefaultTemplate = "HR00"
      &MyPortal.Save();
   End-If;
   ```

You may want to do error checking after you save the PortalRegistry.

Adding a ContentProvider

The following example adds a ContentProvider to an existing PortalRegistry. The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MyPortal;
Local ApiObject &MyCPColl, &MyCProvider;

&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
   /* Do error handling */
End-if;

/* Add a ContentProvider */

&MyCPColl = &MyPortal.ContentProviders;

&MyCProvider = &MyCPColl.InsertItem("HRMS_00");

&MyCProvider.URI = "http://MYMACHINE103100/servlets/iclientservlet/HRMS/";
&MyCProvider.DefaultTemplate = "MYPORTAL_HRMS";
&MyCProvider.Description = "Updated Content Provider for HRMS";

If NOT(&MyPortal.Save()) Then
   &ErrorCol = &MySession.PSMessages;
   For &I = 1 to &ErrorCol.Count
      &Error = &ErrorCol.Item(&I);
   /* do error processing */
   End-For;
&ErrorCol.DeleteAll();
End-If;
```

To add a ContentProvider:

1. Get a Session object and a PortalRegistry.

   Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.) In addition, you want to get a PortalRegistry. Using the GetPortalRegistry method returns a reference to an unpopulated PortalRegistry object.

   ```
   &MyPortal = %Session.GetPortalRegistry();
   ```

2. Open the PortalRegistry.

   After you get a PortalRegistry object, you want to open it, that is, populate it with data. Use the Open method to do this. The Open method returns a Boolean value, and this example uses that value to do error checking to make sure that the PortalRegistry is actually opened. In addition, the name of the PortalRegistry is kept in the record MYRECORD, in the field PORTAL_NAME.

   ```
   If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
      /* Do error handling */
   End-if;
   ```


3. Access the ContentProvider collection.

   You can only add a ContentProvider to a ContentProvider collection. So you must access a ContentProvider collection first, using the ContentProviders property on a PortalRegistry.

```
&MyCPColl = &MyPortal.ContentProviders;
```

4. Add the ContentProvider.

   You must use the name of the ContentProvider with the InsertItem method. This example uses HRMS_00. This method does *not* execute automatically, that is, the ContentProvider isn't actually inserted into the database until the PortalRegistry is saved.

```
&MyCProvider = &MyCPColl.InsertItem("HRMS_00");
```

5. Further define the ContentProvider.

   The URI of a ContentProvider is used in conjunction with content references to define the location of content. So you should define the URI. If a template isn't found for any of the content references on a page at runtime, the system next tries to use the DefaultTemplate defined for a ContentProvider, so you should define a default template.

```
&MyCProvider.URI = "http://MYMACHINE103100/servlets/iclientservlet/HRMS/";
&MyCProvider.DefaultTemplate = "MYPORTAL_HRMS";
&MyCProvider.Description = "Updated Content Provider for HRMS";
```

6. Save the PortalRegistry and check for errors.

   Because there is no Save method with a ContentProvider, you must save the PortalRegistry to complete your changes.

   You can check if there were any errors using the PSMessages property on the session object.

```
If NOT(&MyPortal.Save()) Then
   /* save didn&rsquo;t complete */
   &ErrorCol = &MySession.PSMessages;
   For &I = 1 to &ErrorCol.Count
      &Error = &ErrorCol.Item(&I);
      /* do error processing */
   End-For;
   &ErrorCol.DeleteAll();
End-if;
```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, delete it from the PSMessages collection.

   Note. If you've called the PortalRegistry API an Application Engine program, all errors are also logged in the Application Engine error log tables.

See Chapter 38, "Session Class," Error Handling, page 2180.

Adding a Folder

The following example checks to see if a folder for a user exists. If it doesn't, it creates one.

The folder hierarchy for this example is as follows. The Users folder is where the portal stores things such as user homepages:

```
Root folder

   Users folder

      UserId1 folder

      UserId2 folder

      UserId3 folder

      . . .
```

The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MyPortal;
Local ApiObject &UserFldrColl, &UserFldr;

&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
   /* Do error handling */
End-if;

&UserFldrColl = &MyPortal.RootFolder.Folders.ItemByName("Users").Folders;

&UserFldr = &UserFldrColl.ItemByName(%UserId);

If &UserFldr = Null Then

/* add Folder */

   &UserFldr = &UserFldrColl.InsertItem(%UserId, %UserId);
   &UserFldr.Description = %UserId;

/* Set dates */

   &UserFldr.ValidFrom = %Date;
   &ToDate = AddToDate(%Date, 1, 1, 0);
   &UserFldr.ValidTo = &ToDate;

/* Set properties */

   &UserFldr.PublicAccess = True;

/* save the folder */

   &UserFldr.Save();

End-If;
```

To add a folder:

1. Get a Session object and a PortalRegistry.

   Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.) In addition, you want to get a PortalRegistry. Using the GetPortalRegistry method returns a reference to an unpopulated PortalRegistry object.

   ```
   &MyPortal = %Session.GetPortalRegistry();
   ```


2. Open the PortalRegistry.

   After you get a PortalRegistry object, you want to open it, that is, populate it with data. Use the Open method to do this. The Open method returns a Boolean value, and this example uses that value to do error checking to make sure that the PortalRegistry is actually opened. In addition, the name of the PortalRegistry is kept in the record MYRECORD, in the field PORTAL_NAME.

   ```
   If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
      /* Do error handling */
   End-if;
   ```

3. Access the User folder collection.

   One of the strengths of dot notation is being able to string together many methods and properties into a single line of code. The result of this single line of code is to return a reference to the folder collection within the Users folder.

   ```
   &UserFldrColl = &MyPortal.RootFolder.Folders.ItemByName("Users").Folders;
   ```

   Note that in this kind of code, you must access the RootFolder for the PortalRegistry before you can access the folders collection.

   If there were additional folders in your hierarchy, you could continue in the same way, using ItemByName and Folders.

4. Check if the user already has a folder.

   The system variable %UserId returns the user ID of the current user. Then this example uses the ItemByName method on the folder collection to see if the user already has a folder. This assumes that the user folders are names according to the UserId. The ItemByName method returns a reference to the folder if it exists. If the folder does not exist, ItemByName returns NULL.

   ```
   &UserFldr = &UserFldrColl.ItemByName(%UserId);

   If &UserFldr = Null Then
   ```

5. Add a folder if one doesn't exist.

   Use the UserId as the name of the folder, as well as for the label and the description.

   ```
   &UserFldr = &UserFldrColl.InsertItem(%UserId, %UserId);
   &UserFldr.Description = %UserId;
   ```

6. Set the ValidFrom and ValidTo dates.

   For this example, the folder should be accessible from the date it's created, so the example sets the ValidFrom property to be today's date.

   When you first create a folder, the ValidTo date, that is, the date that this folder "expires", is set to null, that is an empty string. This means it never expires. In this example, we set the ValidTo date to one year and one day after the current date.

   ```
      &UserFldr.ValidFrom = %Date;
      &ToDate = AddToDate(%Date, 1, 1, 0);
      &UserFldr.ValidTo = &ToDate;
   ```


7. Set permissions for the folder.

   There are several properties that work together to determine the access of a folder or content reference. This example sets the PublicAccess property to True, which means that everyone has access to it, and it's included in all collections of folders.

   ```
   &UserFldr.PublicAccess = True;
   ```

   This example sets only the permission properties on the folder. It doesn't set all the possible permissions, such as the PermissionValue objects for the folder.

8. Save the folder.

   After you have finished your changes to the folder, save it.

   ```
   &UserFldr.Save();
   ```

   You may want to check for errors after you save each folder.

See Chapter 30, "Portal Registry Classes," Using Security, page 1591.

Adding a Content Reference

The following code example adds a content reference that's a target component to the Approve Expenses folder.

The folder hierarchy for this example is:

```
Root folder
   Expenses folder
      Create Expenses folder
      Review Expenses folder
      Approve Expenses folder
```

The following is the complete code sample: the steps explain each line.


```
Local ApiObject &MyPortal;
Local ApiObject &CRef, &CRefColl;
Local ApiObject &Fldr;

&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
   /* Do error handling */
End-if;

&Fldr = &MyPortal.RootFolder.Folders.ItemByName("Expenses").Folders.ItemByName⇒
("Approve Expenses");

/* add content reference */

&CRef = &Fldr.ContentRefs.InsertItem(&CRefname, MYCP, MYRECORD.MYURL);
   &CRef.Description = &CRefname;

/* Set dates */
&CRef.ValidFrom = %Date;
&ToDate = AddToDate(%Date, 1, 1, 0);
&CRef.ValidTo = &ToDate;

/* Set content reference  types */
&CRef.UsageType = "TARG";   /* Target content reference */
&CRef.URLType = "UPGE";     /* Component type of content reference */
&CRef.StorageType = "RMTE";  /* Template is remote. */
&CRef.Template = "EXPENSES_TEMPLATE";      /* name of template */
&CRef.TemplateType = "HTML";   /* type of Template */
/* Set URL */
&CRef.URL = "ICType=Panel&Menu=MANAGE_EXPENSES&Market=GBL&Component=APPROVE_FINAL"

/* Save the content reference */

&CRef.Save();
```

To add a content reference:

1. Get a Session object and a PortalRegistry.

   Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.) In addition, you want to get a PortalRegistry. Using the GetPortalRegistry method returns a reference to an unpopulated PortalRegistry object.

   ```
   &MyPortal = %Session.GetPortalRegistry();
   ```

2. Open the PortalRegistry.

   After you get a PortalRegistry object, you want to open it, that is, populate it with data. Use the Open method to do this. The Open method returns a Boolean value, and this example uses that value to do error checking to make sure that the PortalRegistry is actually opened. In addition, the name of the PortalRegistry is kept in the record MYRECORD, in the field PORTAL_NAME.

   ```
   If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
      /* Do error handling */
   End-if;
   ```

3. Access the folder collection.

   One of the strengths of dot notation is being able to string together many methods and properties into a single line of code. The result of this single line of code is to return a reference to the folder collection within the folder.

   ```
   &Fldr = &MyPortal.RootFolder.Folders.ItemByName("Expenses").Folders.ItemByName⇒
   ("Approve Expenses");
   ```

   The Adding a Folder example has more explanation on the break down of this line of code.

4. Add the content reference.

   You can only add content references from the collection of content references for the folder. After you have the content reference collection, use the InsertItem method to add the content reference. The ContentProvider for this content reference is named MYCP. The URL is stored in the record MYRECORD in the field MYURL.

   ```
   &CRef = &Fldr.ContentRefs.InsertItem(&CRefname, MYCP, MYRECORD.MYURL);
   &CRef.Description = &CRefname;
   ```

   Note that the InsertItem method adds the content reference to the database as soon as it's executed. You could check for errors at this point to see if the content reference was added correctly.

5. Set the ValidFrom and ValidTo dates.

   For this example, the content reference should be accessible from the date it's created, so the example sets the ValidFrom property to be today's date.

   When you first create a content reference, the ValidTo date, that is, the date that this content reference "expires", is set to null, that is an empty string. This means it never expires. In this example, we set the ValidTo date to one year and one day after the current date.

   ```
   &CRef.ValidFrom = %Date;
   &ToDate = AddToDate(%Date, 1, 1, 0);
   &CRef.ValidTo = &ToDate;
   ```

6. Set the type parameters.

   Many content reference properties work together to set the type of content reference.

   In this example, the content reference is a target. It's a component type of content reference, which means it's a PeopleSoft definition. The template to be used for displaying this content reference is stored remotely, its name is EXPENSES_TEMPLATE, and it's an HTML template.

   ```
   &CRef.UsageType = "TARG"; /* Target content reference */
   &CRef.URLType = "UPGE"; /* Component type of content reference */
   &CRef.StorageType = "RMTE"; /* Template is remote. */
   &CRef.Template = "EXPENSES_TEMPLATE"; /* name of template */
   &CRef.TemplateType = "HTML"; /* type of Template */
   ```

7. Set the URL.

   The URL property of the content reference indicates where the content actually is. As this content reference is referencing a page, the URL has a specific format to indicate which page.

   ```
   &CRef.URL = "ICType=Panel&Menu=MANAGE_EXPENSES&Market=GBL&Component=APPROVE_FINAL"
   ```


8. Save the content reference.

   After you have finished adding the content reference, you should save it.

   ```
   &CRef.Save();
   ```

   You may want to check for errors after you save each content reference.

See Chapter 30, "Portal Registry Classes," Adding a Folder, page 1811.

Setting Permissions Using the PermissionValue Object

The following code example adds permissions for a folder through the PermissionValue object.

The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MyPortal;
Local ApiObject &UserFldr;
Local ApiObject &PermV, &PermVColl;

&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
   /* Do error handling */
End-if;

&UserFldr = &MyPortal.RootFolder.Folders.ItemByName("Users");

&CascadedColl = &UserFldr.CascadedPermissions;

&PermV = &CascadedColl.ItemByName("VENDOR1");

If NOT &PermV Then
   &PermVColl = &UserFldr.Permissions;
   &PermVColl.InsertItem("VENDOR1");
   &UserFldr.Save();
End-If;
```

To set permissions for a folder, using the PermissionValue object:

1. Get a Session object and a PortalRegistry.

   Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.) In addition, you want to get a PortalRegistry. Using the GetPortalRegistry method returns a reference to an unpopulated PortalRegistry object.

   ```
   &MyPortal = %Session.GetPortalRegistry();
   ```

2. Open the PortalRegistry.

   After you get a PortalRegistry object, you want to open it, that is, populate it with data. Use the Open method to do this. The Open method returns a Boolean value, and this example uses that value to do error checking to make sure that the PortalRegistry is actually opened. In addition, the name of the PortalRegistry is kept in the record MYRECORD, in the field PORTAL_NAME.

   ```
   If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
      /* Do error handling */
   End-if;
   ```

3. Access the User folder.

   One of the strengths of dot notation is being able to string together many methods and properties into a single line of code. The result of this single line of code is to return a reference to the folder named Users.

   ```
   &UserFldr = &MyPortal.RootFolder.Folders.ItemByName("Users");
   ```

4. Access the entire, cascaded PermissionValue collection.

   There are two types of PermissionValue collections you can access for a folder.

   - One contains only the permission list values that are set at that folder. This collection is returned with the Permissions property.
   - The other contains all the permission list values for folder, that is, it contains any permission list values set in any parent folder and cascaded. This collection is returned with the CascadedPermissions property.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This example uses the CascadedPermissions collection to check if the permission exists, because we don't want to add a permission if it already exists. Then it uses the Permissions collection to add the value.

```
&CascadedColl = &UserFldr.CascadedPermissions;
```

1. Check to see if the permission list value exits.

   Before adding the new PermissionValue object, you want to make sure one by that name doesn't already exist.

   ```
   &PermV = &CascadedColl.ItemByName("VENDOR1");
   ```


2. If the PermissionValue doesn't exist, add it.

   The ItemByName returns a reference to a PermissionValue object if it exists, or NULL if it doesn't. So this example checks for NULL, and adds the PermissionValue if it doesn't exist.

   Notice that this example is *not* changing the Cascaded property with the PermissionValue object. The default value for a new PermissionValue object is False. As this example does not want to cascade the permissions for this permission list, it retains the default value.

   In addition, if the PermissionValue object is added, the folder is saved. The InsertItem method executes only after the parent object, the folder, is saved.

```
If NOT &PermV Then
   &PermVColl = &UserFldr.Permissions;
   &PermVColl.InsertItem("VENDOR1");
   &UserFldr.Save();
End-If;
```

   You may want to check for errors after you save the folder.

See Chapter 30, "Portal Registry Classes," Permissions, page 1672 and Chapter 30, "Portal Registry Classes," CascadedPermissions, page 1667.

See Chapter 30, "Portal Registry Classes," Using PermissionValue, RolePermissionValue, Cascading Permissions and CascadingRolePermissions, page 1592.

Using Attributes

The following example uses attributes for a folder to determine what is displayed to an end-user. The end-user still has access to the content, however, it isn't displayed. This program doesn't discuss how to hide or display content: it just shows the function used to determine if a folder should be displayed.

The following is the complete code sample: the steps explain each line.

```
Function checkVisible(&FolderId As ApiObject) Returns Boolean
   &colAttrib = &FolderId.Attributes;

   If &colAttrib.ItemByName("PORTAL_HIDE_FROM_NAV") <> Null Then
      If &colAttrib.ItemByName("PORTAL_HIDE_FROM_NAV").value = "TRUE" Then
             &display = False;
      End-If;
   End-If;

    Return &display;

End-Function;
```

To use attributes for a folder:

1. Access the folder attribute collection.

   A reference to a folder is passed into the function. Then the example uses the Attributes property on the folder to access the attribute collection for the folder.

```
&colAttrib = &FolderId.Attributes;
```


2. Check to see if the folder should be hidden.

   This step is actually two checks. The first check is to see if there is an attribute with the folder called PORTAL_HIDE_FROM_NAV. If the folder has such an attribute, the second check determines the value of this attribute. If both are true, the folder should be hidden, so the variable &display is set to True, and returned.

```
If &colAttrib.ItemByName("PORTAL_HIDE_FROM_NAV") <> Null Then
      If &colAttrib.ItemByName("PORTAL_HIDE_FROM_NAV").value = "TRUE" Then
             &display = False;
      End-If;
   End-If;

    Return &display;
```

Visual Basic Example

The following is an example referencing a PortalRegistry object using Visual Basic.

```
Dim oCref As ContentRef
Dim oCrefColl As ContentRefCollection

Set oSession = CreateObject("PeopleSoft.Session")
iResult = oSession.Connect(1, AppServStr, OperID, OperPasswd, 0)

Set oPortal = oSession.GetPortalRegistry
iResult = oPortal.Open("PORTAL")

Set oCrefColl = oPortal.RootFolder.ContentRefs
Set oCref = oCrefColl.InsertItem("papi012")
iResult = oPortal.Close

iResult = oPortal.Open("PORTAL")

Set oCref = oPortal.FindCRefByName("papi012")

oPortal.Close
oSession.Disconnect
Exit Sub
```

C/C++ Example

The following is a C/C++ example.

```
/*********************************************************************
*                                                                    *
* psportal_test.cpp                                                  *
*                                                                    *
**********************************************************************
*                                                                    *
*              Confidentiality Information:                          *
*                                                                    *
* This module is the confidential and proprietary information of
* PeopleSoft, Inc.; it is not to be copied, reproduced, or
* transmitted in any form, by any means, in whole or in part,
* nor is it to be used for any purpose other than that for which it
* is expressly provided without the written
* permission of PeopleSoft.
*                                                                    *
* Copyright (c) 1988-1999 PeopleSoft, Inc.  All Rights Reserved.     *
*                                                                    *
**********************************************************************
*                                                                    *
* SourceSafe Information:                                            *
*                                                                    *
* $Logfile::                                                        $*
* $Revision::                                                       $*
* $Date::                                                           $*
*                                                                    *
*********************************************************************/
/*********************************************************************
*       includes                                                     *
*********************************************************************/

#ifdef PS_WIN32
#include "stdafx.h"
#endif

#include "bcdef.h"
#include "apiadapterdef.h"
#include "peoplesoft_peoplesoft_i.h"

#include <stdio.h>
#include <iostream.h>
#include <wchar.h>

/*********************************************************************
*       general defines and macros                                   *
*********************************************************************/

/*********************************************************************
*       globals                                                      *
*********************************************************************/
HPSAPI_SESSION                                             hSession;


/*********************************************************************
*       function prototypes                                          *
*********************************************************************/
void DisconnectSession();
void GetFolder(HPSAPI_SESSION hSession, LPTSTR pszPortalName);
void OutError(HPSAPI_SESSION hSession);

/*********************************************************************
* Function:     main                                                 *
*                                                                    *
* Description:                                                       *
*                                                                    *
```



```
* Returns:                                                               *
*************************************************************************/
void main(int argc, char* argv[])
{
    // Declare variables
    PSAPIVARBLOB    blobExtra;
    TCHAR szServer[80] = _T("//LCHE0110:900");
    TCHAR szUserid[30] = _T("PTMO");
    TCHAR szUserPswd[80] = _T("PT");
    TCHAR szPortalName[80] = _T("PORTAL");

    memset(&blobExtra, 0, sizeof(PSAPIVARBLOB));

    // Establish a PeopleSoft Session.
   HPSAPI_SESSION hSession = PeopleSoft_Session_Create();

    if (PeopleSoft_Session_Connect(hSession, 1, szServer, szUserid, szUserPswd,⇒
 blobExtra))
    {
        GetFolder(hSession, szPortalName);
   }
   else
    {
      // Connect to AppServer Failed. Error out.
        if (PeopleSoft_Session_GetErrorPending(hSession))
      {
         wprintf(L"\nConnect to AppServer Failed.\n");
         OutError(hSession);
      }
      else
         wprintf(L"No error pending from session.\n");
    }
    DisconnectSession();
}

/*************************************************************************
*       function implementations                                         *
*************************************************************************/
/*************************************************************************
* Function:     DisconnectSession                                        *
*                                                                        *
* Description:  Disconnects the Session object.                          *
*                                                                        *
* Returns:      None                                                     *
*************************************************************************/
void DisconnectSession()
{
    // Disconnect current session to free memory and session variables.
    if (hSession)
    {
        if (PeopleSoft_Session_Disconnect(hSession))
        {
            PeopleSoft_Session_Release(hSession);
        }
        else
        {
            wprintf(L"Disconnect to AppServer Failed.\n");
        }
    }
}

/*************************************************************************
```

```
* Function:     GetFolder                                                   *
*                                                                           *
* Description:  Get root folder and display folder name                     *
*                                                                           *
* Returns:      None                                                        *
*************************************************************************/
void GetFolder(HPSAPI_SESSION hSession, LPTSTR pszPortalName)
{
    LPTSTR                  szStr;
    HPSAPI_PORTALREGISTRY   hPortal;
    HPSAPI_FOLDER           hRootFolder;

    hPortal=(HPSAPI_PORTALREGISTRY) PeopleSoft_Session_GetPortalRegistry(hSession);
    PortalRegistry_PortalRegistry_Open(hPortal, pszPortalName);
    if (PortalRegistry_PortalRegistry_Open(hPortal, pszPortalName) == false)
        _tprintf(_T("Open portal failed\n"));
    else
    {
        hRootFolder = PortalRegistry_PortalRegistry_GetRootFolder(hPortal);
        szStr = PortalRegistry_Folder_GetName(hRootFolder);
        _tprintf(_T("root folder name = %s\n"), szStr);
    }
}


/*************************************************************************
* Function:     OutError                                                    *
*                                                                           *
* Description:  Output error message from session psMessage object          *
*                                                                           *
* Returns:      None                                                        *
*************************************************************************/
void OutError(HPSAPI_SESSION hSession)
{
    LPTSTR                  szStr;
    HPSAPI_PSMESSAGECOLLECTION hMsgColl;
   HPSAPI_PSMESSAGE hMsg;


    hMsgColl=(HPSAPI_PSMESSAGECOLLECTION) PeopleSoft_Session_GetPSMessages(h⇒
Session);

   wprintf(L"Get psMessage collection ok.\n");

   int i;
   PSI32 count;
   count=(PSI32) PeopleSoft_PSMessageCollection_GetCount;
   wprintf(L"Count= %d\n", count);

   for (i=0; i<count; i++)
   {
      hMsg=(HPSAPI_PSMESSAGE) PeopleSoft_PSMessageCollection_Item(hMsgColl, i);
      szStr=PeopleSoft_PSMessage_GetText(hMsg);
      wprintf(L"\nMessage from session: %s\n",szStr);
   }

}
```


Chapter 31

PostReport Class

This chapter provides an overview of PostReport class and discusses the following topics:

- Data type of a PostReport object
- Scope of a PostReport object
- Determining the distribution list
- PostReport class built-in functions
- PostReport class properties
- PostReport examples

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Using Report Manager"

PostReport Class Overview

The PostReport class provides properties and method for posting reports to the Report Repository. This class enables you to create reports outside of Process Scheduler and have them posted in the Report Repository.

The request to post reports creates an entry to the Report Manager table. The Distribution Agent polls the Report Manager table for any new requests and proceeds with any request it finds by transferring the files to the repository server. The report can then be accessed from the Report Manager after the Distribution Agent successfully processes the post request.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Using Report Manager"

Determining the Distribution List

Use the AddDistributionOption method to authorize users to view the report from the Report Manager. This function enables you to grant access to the report either specifying a User ID or Role. However, if no distribution list is specified in the request, the class queries the system for the distribution list as follows:

1. If the Process Instance is specified, the system checks for the distribution list in the Process Request table. The system assumes in this case that the request was scheduled through the Process Request Dialog, but was processed by a third-party scheduler.

2. If a Process Name and Process Type properties were specified, the system checks if a distribution list was predefined in the Process Definition table.

3. If no list was found using the above queries, the user who submitted the request has sole access to the report.

Data Type of a PostReport Object

A PostReport object is declared using the PostReport data type. For example:

```
Local PostReport &POSTRQST;
```

Scope of a PostReport Object

A PostReport object can only be instantiated from PeopleCode. However, if you need to access the Process Schedule Manger from outside of PeopleCode, like in a Visual Basic program, you can use the Process Schedule Manager API.

This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Understanding PeopleSoft Process Scheduler"

PostReport Class Built-in Function

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SetPostReport

PostReport Class Methods

In this section, we discuss the PostReport class methods. The methods are discussed in alphabetical order.

AddDistributionOption

Syntax

AddDistributionOption*(DistIdType,DistId)*

Description

Use the AddDistributionOption method to specify the users authorized to view the report once it is available in the Report Manager.

Parameters

Parameter	***Description***
DistIdType	Specify the distribution ID type as a string. This identifies if the value passed in the DistID is either a user or role. Values for this parameter are: • User • Role
DistID	Specify the distribution ID as a string.

Returns

Returns a number: 0 if successful, 1 if the system detected an error in the parameter passed.

Example

The following example grants access to the VP1 user and all users associated with the MANAGERS role.

```
&POSTRQST.AddDistributionOption("User", "QEDMO" );
&POSTRQST.AddDistributionOption("Role", "MANAGERS");
```

Put

Syntax

Put()

Description

The Put method inserts a request in the Report Manager table which runs according to the values in the properties of the PostReport object. In order to successfully schedule a process or job, certain properties are required.

Parameters

None.

Returns

None. If you want to verify that the method executed successfully, check the value of the Status property.

Example

```
&POSTRQST.Put();
&RPTINSTANCE = &POSTRQST.ReportId;
If (&RPTINSTANCE > 0) Then
      MessageBox(0, "", 63, 119, "Successfully processed request with Rpt. ID %1⇒
 for Process %2 to post from directory %3", &RPTINSTANCE, &POSTRQST.ProcessName,⇒
 &POSTRQST.SourceReportPath);
   Else
      MessageBox(0, "", 63, 122, "Not successful for process request for Process⇒
 %1 to post from directory %2", &POSTRQST.ProcessName, &POSTRQST.SourceReportPath);
End-If;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SetPostReport

PostReport Class Properties

In this section, we discuss the PostReport class properties. The properties are discussed in alphabetical order.

ExpirationDate

Description

This property specifies the date the report will be deleted from the Report Manager table. If the expiration date is not specified, the system calculates the expiration date based on the Retention Days specified in the Process Scheduler's System Settings.

This property is read-write.

Example

```
&MYRQST.ExpirationDate = "2003/01/01";
```

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Defining PeopleSoft Process Scheduler Support Information," Defining System Settings

OutDestFormat

Description

This property specifies the output format of the file that is posted to the Report Repository. If format is not specified, the system determines the format based on the extension of the file being posted to the Report Repository.

This property is read-write.

Example

```
&MYRQST.OutDestFormat = "PDF";
```

See Also

Chapter 32, "Process Request Classes," Values for Output Type and Format, page 1841

ProcessInstance

Description

This property specifies the instance number assigned to request. This property is only required if the report was initially submitted through the Process Request Dialog, but was processed by a third-party scheduler.

This property is read-write.

See Also

Chapter 31, "PostReport Class," Determining the Distribution List, page 1825

ProcessName

Description

This property specifies the name of a predefined process as a string.

This property is read-write.

Example

```
&MYRQST.ProcessName = "XRFWIN";
```

See Also

Chapter 31, "PostReport Class," ProcessType, page 1830

Chapter 32, "Process Request Classes," RunControlID, page 1874

ProcessType

Description

This property specifies the name of a predefined process type as a string.

This property is read-write.

Example

Note that spaces are included in the string for ProcessType.

```
&MYRQST.ProcessType = "Application Engine";
```

See Also

Chapter 31, "PostReport Class," ProcessName, page 1830

Chapter 32, "Process Request Classes," RunControlID, page 1874

ReportDescr

Description

This property specifies the description as a string.

If the ReportDescr property is not specified, and both Process Name and Process Type are not null, the description is extracted from the Process Definition table.

This property is read-write.

Example

```
&MYRQST.ReportDescr = "Panel Cross Reference Report";
```

ReportFolder

Description

This property specifies the name of the report folder as a string. If you specify a folder, the folder that you specify must have already been defined using Process Scheduler's Report Folders Administration.

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Using Report Manager," Understanding Report Folders

ReportId

Description

This property is a system-generated identification number. The system assigns a ReportId after the Put method processed the request.

This property is read-write.

Example

```
&MYRQST.Put();
if &MYRQST.ReportId > 0 then
     MessageBox(o, "", 63, 119, "Sucessfully processed request with Rpt. ID %1",⇒
 &MYRQST.ReportId);
Else
    /* do error processing */
End-If;
```


ServerName

Description

This property specifies the Process Scheduler Server name that posts the report. This property takes a string value.

This property is read-write.

Example

```
&MYRQST.ServerName = "PSNT";
```

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Defining PeopleSoft Process Scheduler Support Information," Defining System Settings

SourceReportPath

Description

This property specifies the path that contains the source report. You need to specify an absolute path.

This property is read-write.

PostReport Class Examples

The following example posts files found in the directory 'c:\temp\SQRDIR' and can be accessed by VP1 and users in the MANAGERS role.


```
Local PostReport &RPTINFO;
   Local number &RPTINSTANCE

   /*********************************************************************
   * Construct a PostReport Object.                                    *
   *********************************************************************/
   &RPTINFO = SetPostReport();

   &RPTINFO.ProcessName = "GLS7009";
   &RPTINFO.ProcessType = "SQR Report";
   &RPTINFO.ReportFolder = "Financial";
   &RPTINFO.SourceReportPath = "c:\temp\SQRDIR";
   &RPTINFO.ExpirationDate = "2005/01/01";
   &RPTINFO.ReportDescr = "Journal Posting Summary Report";
   &RPTINFO.ServerName = "PSNT";

   &RPTINFO.AddDistributionOption("USER", "VP1");
   &RPTINFO.AddDistributionOption("ROLE", "MANAGERS");

   &RPTINFO.Put();
   &RPTINSTANCE = &RPTINFO.ReportId;

   If (&RPTINSTANCE > 0) Then
      MessageBox(0, "", 63, 119, "Successfully processed request with Rpt. ID %1⇒
 for Process %2 to post from directory %3", &RPTINSTANCE, &RPTINFO.ProcessName,⇒
 &RPTINFO.SourceReportPath);
   Else
      MessageBox(0, "", 63, 122, "Not successful for process request for Process⇒
 %1 to post from directory %2", &RPTINFO.ProcessName, &RPTINFO.SourceReportPath);
   End-If;
```

Chapter 32

Process Request Classes

This chapter provides an overview of the process request classes and discusses the following topics:

- Data type of a ProcessRequest object.
- Scope of a ProcessRequest object.
- Options for items in jobs and jobsets.
- Values for output type and format.
- Alternative option to specify email or web attributes.
- ProcessRequest class built-in function.
- ProcessRequest class methods.
- ProcessRequest class properties.
- ProcessRequest examples.
- PrcsApi class.
- Scope of a PrcsApi object.
- Data type of a PrcsApi object.
- How to import the PrcsApi class.
- How to create a PrcsApi object.
- PrcsApi class constructor.
- PrcsApi class methods.

Understanding Process Request Classes

The ProcessRequest class provides properties and a method for scheduling a pre-defined process or job. You must define the process or the job (using Process Scheduler Manager) *before* you can schedule it using the ProcessRequest class.

The properties of this class contain the same values as those required by Process Scheduler Manager for scheduling a process or job. Values you provide for these properties may override the equivalent values set in Process Scheduler Manager, depending on the override settings you make in PeopleSoft Process Scheduler pages.

Starting with PeopleSoft 8.4, PeopleSoft supports only the ProcessRequest class to schedule a process or a job in PeopleCode. The ScheduleProcess PeopleCode function is no longer supported. Any existing PeopleCode functions containing this deprecated function must be modified to use the ProcessRequest class to schedule a request.

This chapter assumes that the reader has working knowledge of PeopleSoft Process Scheduler.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler

Data Type of a ProcessRequest Object

A ProcessRequest object is declared using the ProcessRequest data type. For example:

```
Local ProcessRequest &RQST;
```

Scope of a ProcessRequest Object

A ProcessRequest object can be instantiated only from PeopleCode. However, if you need to access the Process Schedule Manager from outside of PeopleCode, for example, in a Visual Basic program, you can use the Process Schedule Manager API.

This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler

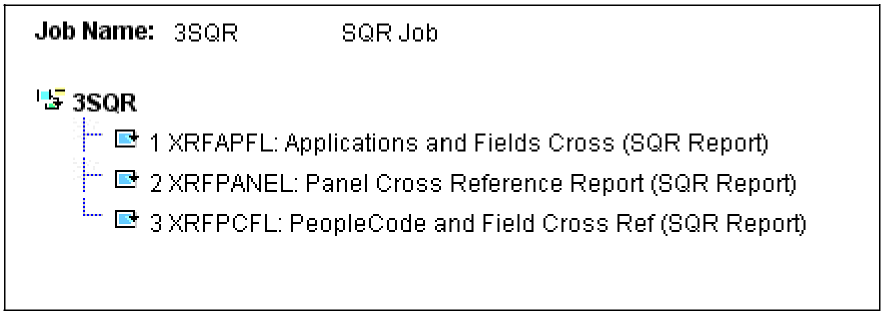
Options for Items In Jobs and Jobsets

A key feature of PeopleSoft Process Scheduler is the option to create a Job Definition consisting of other jobs. A job containing a job is called a *jobset.* Process Scheduler Server schedules the items within these jobs according to how the jobset is defined through the Process Scheduler Manager.

The ProcessRequest class requires the following properties to handle jobsets:

- JobName: indicates the name of the job that an item belongs to.
- PrcsItemLevel: indicates a job item's process item level within a jobset.
- JobSeqNo: indicates a job item's job sequence number within the Process Item Level.

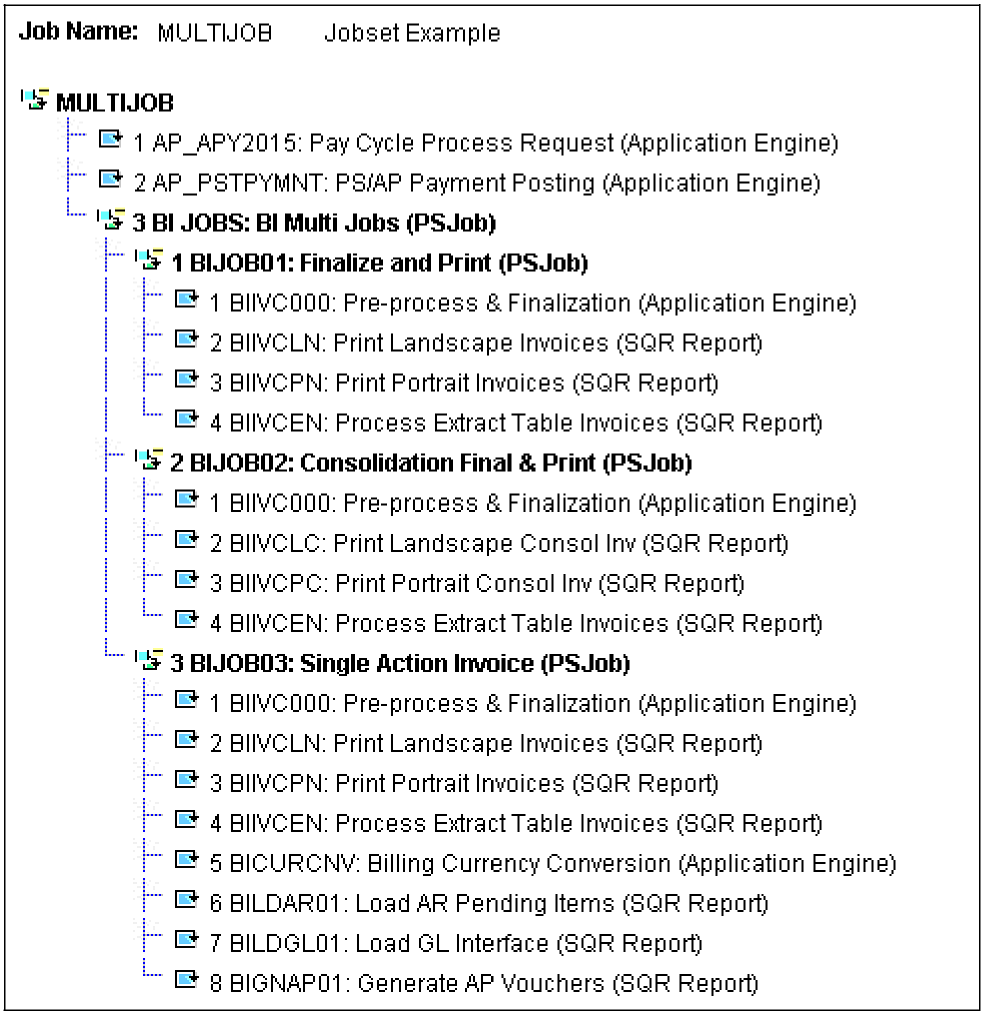
The following is a simple example of a job containing only a single process:



Sample job with single process

Process	***PrcsItemLevel***	***JobName***	***JobSeqNo***
3SQR	0		0
XRFAPFL	1	3SQR	1
XRFPANEL	2	3SQR	2
XRFPCFL	3	3SQR	3

The following is an example of a jobset containing multiple jobs:



Sample jobset containing multiple jobs

Process	***PrcsItemLevel***	***JobName***	***JobSeqNo***
MULTIJOB	0		0
AP_APY2015	1	MULTIJOB	1
AP_PSTPYMNT	1	MULTIJOB	2
BI JOBS	1	MULTIJOB	3
BIJOB01	2	BI JOBS	1
BIIVC000	3	BIJOB01	1

Process	***PrcsItemLevel***	***JobName***	***JobSeqNo***
BIIVCLN	3	BIJOB01	2
BIIVCPN	3	BIJOB01	3
BIIVCEN	3	BIJOB01	4
BIJOB02	2	MULTIJOB	2
BIIVC000	3	BIJOB02	1
BIIVCLC	3	BIJOB02	2
BIIVCPC	3	BIJOB02	3
BIIVCEN	3	BIJOB02	4
BIJOB03	2	MULTIJOB	3
BIIVC000	3	BIJOB03	1
BIIVCLN	3	BIJOB03	2
BIIVCPN	3	BIJOB03	3
BIIVCEN	3	BIJOB03	4
BICURCNV	3	BIJOB03	5
BILDAR01	3	BIJOB03	6
BILDGL01	3	BIJOB03	7
BIGNAP01	3	BIJOB03	8

All the class methods that are used to change options for items within a job or jobset, use these attributes as optional parameters.

If a method is used without specifying any of these parameters, the system assumes the changes apply to *all* items within a jobset.

If one or more of these attributes are specified, the method applies the changes to items matching the value of these attributes.

In the case where a JobName is specified, the changes apply to all items within the specified job.

Specific methods used to manipulate options within a job or jobset include:

- AddDistributionOption
- SetEmailOption
- SetItemFolder
- SetOutputOption

When manipulating the options within a job or jobset, you must specify the process type and process name in the CreateProcessRequest function when instantiating a ProcessRequest object. This is required as the class needs to retrieve the job item information from the job definition for a specific job or jobset. The class uses this information to alter items within the job or jobset based on subsequent method calls made prior to scheduling the request. If the process type and process name are not specified when instantiating the ProcessRequest object, all PeopleCode programs trying to use any of the methods return an error message:

```
The method can not be used because the ProcessRequest is not properly initialized
```

In the previous jobset example, the ProcessRequest must be instantiated as follows:

```
Local ProcessRequest &RQST;

&RQST = CreateProcessRequest("PSJob", "MULTIJOB");
```

The following code examples use the previous example MULTIJOB, and show how items can be altered within a job or jobset:

- Change all items in the Jobset to "Web" with reports generated in PDF format:

  ```
  &RQST.SetOutputOption("Web", "PDF", "", "MULTIJOB");
  ```

  Alternatively, this can also be coded as follows:

  ```
  &RQST.SetOutputOption("Web", "PDF", "");
  ```

- Change the output option for job BIJOB02 to "Email" with reports generated in HTM format:

  ```
  &RQST.SetOutputOption("Email", "HTM", ", "BIJOB02");
  ```

- Change the output option for BIIVCPN in job BIJOB01 to "File" in LP (LinePrinter) format:

  ```
  &RQST.SetOutputOption("File", "LP", "BIJOB01", 3, 3);
  ```

- Change all items in level three to "Web" in PS (PostScript) format:

  ```
  &RQST.SetOutputOption("Web", "PS", "", "", 3);
  ```


Values for Output Type and Format

The values for output type and format are based on the generic process type assigned to a process type definition. The following table provides a cross-reference listing for all delivered process type definitions.

Generic Process Type	***Process Type***
AppEngine	Application Engine, Optimization Engine
COBOL	COBOL SQL
Crystal	Crystal, Crystal Check
Cube	CubeBuilder, HyperCube Builder
SQR	SQR Process, SQR Report
Winword	Winword
nVision	nVision, nVision-Report, nVision-ReportBook
Data Mover	Data Mover
Other	Essbase

The following table lists all the values for OutDestType delivered with PeopleTools.

However, the values and defaults can be customized through Process Scheduler Manager. Please refer to the PeopleSoft Process Scheduler documentation for additional information on how to accurately determine the values for your system.

Generic ProcessType	***Valid OutDestTypes***	***Defaults***
AppEngine	FILE, WEB, WINDOW	WEB
COBOL	NONE, WINDOW, WEB	NONE
Crystal	WEB, WINDOW, EMAIL, FILE, PRINTER	WEB

Generic ProcessType	***Valid OutDestTypes***	***Defaults***
Cube	NONE	NONE
nVision	WEB, WINDOW, EMAIL, FILE, PRINTER, DEFAULT	DEFAULT
SQR	WEB, WINDOW, EMAIL, FILE, PRINTER	WEB
WinWord	WINDOW, WEB	WEB
Data Mover	WEB,WINDOW, FILE	WEB
OTHER	WEB, WINDOW, EMAIL, FILE, PRINTER, NONE	NONE

Similar to the OutDestType, the formats and defaults for a generic process type can be customized through Process Scheduler Manager. The following table lists the values and defaults as delivered.

Generic ProcessType	***OutDestType***	***Valid OutDestFormats***	***Default***
AppEngine	WINDOW	PDF, XLS, TXT, HTM	TXT
AppEngine	FILE	PDF, XLS, TXT, HTM	TXT
AppEngine	WEB	PDF, XLS, TXT, HTM	TXT
COBOL	NONE	NONE	NONE
COBOL	WEB	TXT	TXT
COBOL	WINDOW	TXT	TXT
Crystal	PRINTER	RPT	RPT
Crystal	FILE	RPT, RTF, TXT, PDF, HTM, XLS, DOC	PDF
Crystal	WINDOW	RPT, RTF, TXT, PDF, HTM, XLS, DOC	PDF

Generic ProcessType	*OutDestType*	*Valid OutDestFormats*	*Default*
Crystal	WEB	RPT, RTF, TXT, PDF, HTM, XLS, DOC	PDF
Crystal	EMAIL	RPT, RTF, TXT, PDF, HTM, XLS, DOC	PDF
Cube	NONE	NONE	NONE
Data Mover	FILE	TXT	TXT
Data Mover	WINDOW	TXT	TXT
Data Mover	WEB	TXT	TXT
nVision	EMAIL	HTM, XLS	XLS
nVision	FILE	HTM, XLS	XLS
nVision	PRINTER	HTM, XLS	XLS
nVision	WINDOW	HTM, XLS	XLS
nVision	WEB	HTM, XLS	XLS
nVision	DEFAULT	DEFAULT	DEFAULT
SQR	EMAIL	CSV, HP, HTM, LP, PDF, PS, SPF,OTHER	PDF
SQR	FILE	CSV, HP, HTM, LP, PDF, PS, SPF,OTHER	PDF
SQR	PRINTER	HP, LP, PS, WP	PS
SQR	WEB	CSV, HP, HTM, LP, PDF, PS, SPF,OTHER	WEB

Generic ProcessType	OutDestType	Valid OutDestFormats	Default
SQR	WINDOW	CSV, HP, HTM, LP, PDF, PS, SPF,OTHER	WEB
WinWord	NONE	NONE	NONE
WinWord	WEB	DOC	DOC
WinWord	WINDOW	DOC	DOC
OTHER	NONE	NONE	NONE

Alternative Options to Specify Email or Web Attributes

To set the email or web attributes in PeopleTools versions prior to release 8.4, your PeopleCode program was similar to the following:

To Set Attributes for Web (Prior to 8.4)

The following example shows how to set web attributes in PeopleTools versions prior to release 8.4:

```
&RQST.OutDestType = "Web";
&RQST.OutDestFormat = "PDF";
&RQST.OutDest = "User : VP1,Role :Managers";
```

Set Attributes for Email (Prior to 8.4)

The following example shows how to set email attributes in PeopleTools versions prior to release 8.4:

```
Local string &Subject ;
Local string &Text;

&Subject = "SQR Report:  Cross Reference Listing";
&Text = "This text will be displayed as the text of this email ";
&RQST.OutDestType = "Email";
&RQST.OutDestFormat = "PDF";
&RQST.OutDest = "User : VP1,Role : MANAGERS";
&RQST.EmailSubject = &Subject;
&RQST.EmailText = &Text;
&RQST.EmailAttachLog = False;
```

Set Attribute for Web (8.4 and Later)

The following example shows how to set web attributes in PeopleTools versions release 8.4 and later:


```
&RQST.SetOutputOption("Web", "PDF", "");
&RQST.AddDistributionOption("User", "QEDMO");
&RQST.AddDistributionOption("Role", "MANAGERS");
```

Set Attribute for Email (8.4 and Later)

The following example shows how to set web attributes in PeopleTools versions release 8.4 and later:

```
Local string &Subject ;
Local string &Text;

&Subject = "SQR Report:  Cross Reference Listing";
&Text = "This text will be displayed as the text of this email ";
&RQST.SetOutputOption("Email", "PDF", "");
&RQST.SetEmailOption(&Subject, &Text, "abc@xyz.com", "False", "False");
&RQST.AddDistributionOption("User", "QEDMO");
&RQST.AddDistributionOption("Role", "MANAGERS");
```

File Dependant Processing

You can define a process to be file dependent, which means associating a file with a process that gets scheduled once the system detects the presence of the file.

You can use the PrcsApi class methods for detecting files associated with a process, as well as for displaying additional messages about the process as it's running.

The PrcsApi class is an Application Package class that you must import into your PeopleCode in order to use its methods.

For example, you could use the PrcsApi class with a Application Engine program. In the Application Engine process PROCESS_EDI, you could create a file dependency (using the FileName ProcessRequest property) and make the process dependent on the file /vendor/edi*.data.

On 10/23, the system detects the file /vendor/edi_1023.data and schedules the process PROCESS_EDI. In the Application Engine program, you can refer to the file that started the process using the getAllFileNames PrcsApi class method.

On 10/24, the system detects the file /vendor/edi_1024.data and schedules the process PROCESS_EDI. The process is started from a different file, and can access that file, so duplicate processing doesn't occur.

See Also

Chapter 32, "Process Request Classes," PrcsApi Class, page 1880

ProcessRequest Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateProcessRequest

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetNextProcessInstance

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SetupScheduleDefnItem

ProcessRequest Class Methods

In this section, we discuss the ProcessRequest class methods. The methods are discussed in alphabetical order.

AddDistributionOption

Syntax

```
AddDistributionOption(DistIdType,DistId [, JobName] [, PrcsItemLevel] [,
JobSeqNo] [, ItemJobSeq])
```

Description

Use the AddDistributionOption method to set the distribution options for any job item in the main job. Distribution options enable you to distribute output in different formats (HTML, PDF, Excel, and so on) to other users based on their user ID or role ID.

This function is valid only if the output destination for the request is routed either to Web or Email.

Parameters

Parameter	***Description***
DistIdType	Specify the distribution ID type as a string. This identifies if the value passed in the DistID is either a user or role. Values for this parameter are: • User • Role
DistID	Specify the distribution ID as a string.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.
JobSeqNo	Specify the job item's job sequence number within the process item level as a number.
ItemJobSeq	Specify the job item's sequence within its parent job as a number

Returns

Returns a number: 0 if successful, 1 if system detected an error in the parameter passed.

Example

The following example grants access to the QEDMO user all reports in MULTIJOB from the Web, while users with role of MANAGERS have access to reports created in the BIJOB03 job.

```
&RQST.SetOutputOption("Web", "PDF", "");
&RQST.AddDistributionOption("User", "QEDMO", "MULTIJOB" );
&RQST.AddDistributionOption("Role", "MANAGERS", "BIJOB03");
```

See Also

Chapter 32, "Process Request Classes," SetEmailOption, page 1857 and Chapter 32, "Process Request Classes," SetOutputOption, page 1863

AddNotifyInfo

Syntax

`AddNotifyInfo(field_name,field_value [, JobName] [, PrcsItemLevel] [, JobSeqNo])`

Description

Use the AddNotifyInfo method to specify name-value pair data to be included in the process status notification message.

Parameters

Parameter	*Description*
field_name	Specify the field name data as a string.
field_value	Specify the field value data as a string.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.
JobSeqNo	Specify the job item's job sequence number within the process item level as a number.

Returns

None.

Example

```
&RQST.AddNotifyInfo("AEMINITEST Name", "Status");
&RQST.AddNotifyInfo("AEMINITEST Descr", "Status Notify");
&RQST.AddNotifyInfo("AEMINITEST OutType", "EMAIL");
```

See Also

Chapter 32, "Process Request Classes," SetNotifyAppMethod, page 1859 and Chapter 32, "Process Request Classes," SetNotifyService, page 1862

PrintJobHTMLRpt

Syntax

PrintJobHTMLRpt()

Description

Use the PrintJobHTMLRpt method to generate an HTML report file displaying all items in a job or jobset in a tree as defined in the Job Definition component.

Parameters

None.

Returns

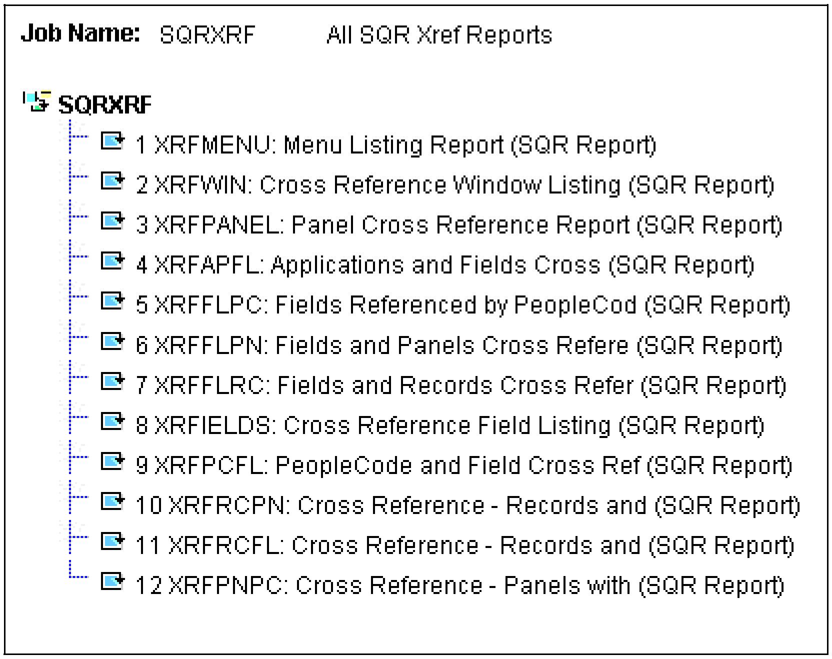
An HTML report as a string

Example

The following code:

```
Local ProcessRequest &JobRQST;
Local string &sHTML;

&JobRQST = CreateProcessRequest("PSJob", "SQRXRF");
&sHTML = &JobRQST.PrintJobHTMLRpt();
```

generates the following HTML file:



Example HTML file

See Also

Chapter 32, "Process Request Classes," PrintJobRqstRpt, page 1849 and Chapter 32, "Process Request Classes," PrintSchdlHTMLRpt, page 1852

PrintJobRqstRpt

Syntax

```
PrintJobRqstRpt(ProcessInstance,ItemInstance [, PrintJobTree] [, PrintDistList]
[, PrintNotifyList] [, PrintSystemMessage] [, PrintApplicationMessage] [,
PrintParamList])
```

Description

Use the PrintJobRqstRpt method to generate an HTML report file displaying the current status of a specific process, job, or jobset.

Parameters

Parameter	*Description*
ProcessInstance	Specify the ProcessInstance number assigned to the process, job, or jobset

Parameter	***Description***
ItemInstance	Specify either to display all items in a job or jobset, or just a specific item. • To display all items, specify 0. • To display a specific item, specify the ProcessInstance number assigned to that item. For a process, set this parameter to 0.
PrintJobTree	Specify whether to have the job tree displayed in the HTML report. This parameter takes a string value: • "1" to display the tree. • "0" to not display the tree. Default value is "1".
PrintDistList	Specify whether you want the list of Users and Roles who will be the recipient of a report generated for a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the list. • "0" to not display the list. Default value is "0".
PrintSystemMessage	Specify whether you want to have the message specified from the Process Definition or Job Definition page displayed in the HTML report. This parameter takes a string value: • "1" to display the system message. • "0" to not display the system message. Default value is "0".
PrintApplicationMessage	Specify whether you want the application messages displayed in the HTML report. These are the application messages that can be viewed from the Message Log subpage of the Process Monitor Detail page. This parameter takes a string value: • "1" to display application messages. • "0" to not display application messages. Default value is "0".
PrintParamList	Specify whether you want the parameter list for a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the parameter list. • "0" to display the parameter list. Default value is "0".

Returns

An HTML report as a string.

Example

The following PeopleCode program:

```
Local ProcessRequest &JobRQST;
Local string &sHTML;
Local string &sPRINT_JOBTREE;
Local string &sPRINT_DISTLIST;
Local string &sPRINT_SYSMESSAGE;
Local string &sPRINT_APPLMESSAGE;
Local string &sPRINT_PARAMLIST;

&sPRINT_JOBTREE = "0";
&sPRINT_DISTLIST = "1";
&sPRINT_SYSMESSAGE = "1";
&sPRINT_APPLMESSAGE = "1";
&sPRINT_PARAMLIST = "1";
&JobRQST = CreateProcessRequest();
&sHTML = &JobRQST.PrintJobRqstRpt(PMN_PRCSLIST.PRCSINSTANCE, 0, &sPRINT_JOBTREE,⇒
 &sPRINT_DISTLIST, &sPRINT_SYSMESSAGE, &sPRINT_APPLMESSAGE, &sPRINT_PARAMLIST);
```

Creates the following HTML report:

Job Name: 3MIX - Mix Jobs

Mode: Parallel

Seq.	Instance	Process Name	Description	Process Type	Run Status	Run Control ID	Type	Output Format	Server Name	Begin Date/Time	End Date/Time
1	4200	XRFWIN	Cross Reference Window Listing	SQR Report	Error	TEST	Web	Acrobat (*.pdf)	PSNT2		2002 - 01 - 08 - 04.01.33.687000

System Message:

Contact John Smith at (555)999-1234 to review the errors

Parameter:

C:\pt840ic3\BIN\SERVER\WINX86\PSSQR.EXE -CT MICROSFT -CS -CD FS840IC2 -CA ACCESSID -CAP ACCESSPSWD -RP XRFWIN -I 4200 -R TEST -CO VP1 -OT 6 -OP C:\pt840ic3\APPSERV\prcs\FS840IC2\log_output\SQR_XRFWIN_4200 -OF 2

Distribution List:

Type	Name
User	VP1

Application Messages:

Process Request shows status of 'INITIATED' or 'PROCESSING' but no longer running (65,70)

Seq.	Instance	Process Name	Description	Process Type	Run Status	Run Control ID	Type	Output Format	Server Name	Begin Date/Time	End Date/Time
2	4201	AEMINITEST	Simple AE test program	Application Engine	Success	TEST	Web	Text Files (*.txt)	PSNT2	2002 - 01 - 08 -	

Example HTML report

See Also

Chapter 32, "Process Request Classes," PrintJobHTMLRpt, page 1848 and Chapter 32, "Process Request Classes," PrintSchdlHTMLRpt, page 1852

PrintSchdlHTMLRpt

Syntax

```
PrintSchdlHTMLRpt([PrintJobTree] [, PrintDistList] [, PrintNotifyList] [,
PrintMessageList] [, PrintParamList])
```

Description

Use the PrintSchdlHTMLRpt method to generate an HTML report file displaying all items in a job or jobset as defined in the Scheduled Jobset Definition component.

Parameters

Parameter	*Description*
PrintJobTree	Specify whether to have the job tree displayed in the HTML report. This parameter takes a string value: • "1" to display the tree. • "0" to not display the tree. Default value is "1".
PrintDistList	Specify whether you want the list of Users and Roles who will be the recipient of a report generated for a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the list. • "0" to not display the list. Default value is "0".
PrintNotifyList	Specify whether you want the list of Users and Roles who will be notified for the status of a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the list. • "0" to not display the list. Default value is "0".
PrintMessageList	Specify whether you want the messages that will be emailed upon completion of a job item displayed in the HTML report. This parameter takes a string value: • "1" to display messages. • "0" to not display messages. Default value is "0".
PrintParamList	Specify whether you want the parameter list for a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the parameter list. • "0" to display the parameter list. Default value is "0".

Returns

An HTML report as a string

Example

The following code:

```
Local string &sHTML;
Local string &sPRINT_JOBTREE;
Local string &sPRINT_DISTLIST;
Local string &sPRINT_NOTIFYLIST;
Local string &sPRINT_MESSAGELIST;
Local string &sPRINT_PARAMLIST;

&sPRINT_JOBTREE = "1";
&sPRINT_DISTLIST = "0";
&sPRINT_NOTIFYLIST = "0";
&sPRINT_MESSAGELIST = "0";
&sPRINT_PARAMLIST = "0";
&JobRQST = SetupScheduleDefnItem("Sample", "MULTIJOB");
&sHTML = &JobRQST.PrintSchdlHTMLRpt(&sPRINT_JOBTREE, &sPRINT_DISTLIST, &sPRINT_⇒
NOTIFYLIST, &sPRINT_MESSAGELIST, &sPRINT_PARAMLIST);
```

Creates the following HTML report:

Job Name: MULTIJOB - Jobset Example

Mode: Serial

Seq.	Process Name	Description	Process Type	Run Control ID	Type	Output Format	Destination	Server Option	Run Time
1	AP_APY2015	Pay Cycle Process Request	Application Engine	RunCntlTest	Web	Text Files (*.txt)	Distribution List	Any Server	
2	AP_PSTPYMNT	PS/AP Payment Posting	Application Engine	RunCntlTest	Web	Text Files (*.txt)	Distribution List	Any Server	
3	BI JOBS	BI Multi Jobs	PSJob	RunCntlTest	(None)	(None)		Any Server	

Job Name: BI JOBS - BI Multi Jobs

Mode: Serial **Parent Job Name: MULTIJOB**

Seq.	Process Name	Description	Process Type	Run Control ID	Type	Output Format	Destination	Server Option	Run Time
1	BIJOB01	Finalize and Print	PSJob	RunCntlTest	(None)	(None)		Any Server	
2	BIJOB02	Consolidation Final & Print	PSJob	RunCntlTest	(None)	(None)		Any Server	
3	BIJOB03	Single Action Invoice	PSJob	RunCntlTest	(None)	(None)		Any Server	

Example HTML report

See Also

Chapter 32, "Process Request Classes," PrintJobHTMLRpt, page 1848 and Chapter 32, "Process Request Classes," PrintJobRqstRpt, page 1849

RunJobSetNow

Syntax

RunJobSetNow()

Description

Use the RunJobSetNow method to schedule a jobset based on settings defined in the Scheduled Jobset Definition component.

Parameters

None.

Returns

None. To verify that the method executed successfully, check the value of the Status property.

Example

```
/* Create the ProcessRequest Object & Run the Scheduled JobSet Now */
Local ProcessRequest &JobRQST;
Local integer &instanceList;

&JobRQST = SetupScheduleDefnItem("Sample","MULTIJOB");
&JobRQST.RunJobSetNow();
&instanceList = &JobRQST.ProcessInstance;
```

See Also

Chapter 32, "Process Request Classes," Status, page 1877

Schedule

Syntax

Schedule()

Description

Use the Schedule method to insert a request in the Process Request table which runs according to the values in the properties of the ProcessRequest object. To successfully schedule a process or job, certain properties are required.

- If you're scheduling a single process, you must assign values to the following properties:
    - RunControlID
    - ProcessName
    - ProcessType
- If you're scheduling a job where job item changes are not made, you must assign values to the following properties:
    - RunControlID
    - JobName
- If you're scheduling a job where job item changes are made, you must assign values to the RunControlID.

Note. You don't have to assign the JobName in this instance because it's set with the CreateProcessRequest function.

Parameters

None.

Returns

None. To verify that the method executed successfully, check the value of the Status property.

Example

```
&MYRQST.Schedule();
If &MYRQST.Status = 0 then
   /* Schedule succeeded. */
Else
   /* Process (job) not scheduled, do error processing */
End-If;
```


See Also

Chapter 32, "Process Request Classes," RunControlID, page 1874; Chapter 32, "Process Request Classes," JobName, page 1868; Chapter 32, "Process Request Classes," ProcessName, page 1872; Chapter 32, "Process Request Classes," ProcessType, page 1873 and Chapter 32, "Process Request Classes," Status, page 1877

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateProcessRequest

SetEmailOption

Syntax

```
SetEmailOption(EmailSubject,EmailText,EmailAddress,EmailWebReport,EmailAttachLog
 [, JobName] [, PrcsItemLevel] [, JobSeqNo] [, ItemJobSeq])
```

Description

Use the SetEmailOption method to set the email options for all job items in the main job.

Parameters

Parameter	***Description***
EmailSubject	Specify the text used in the subject of the email sent at completion of this job email subject as a string. You can specify a Null ("") for this parameter.
EmailText	Specify the email text sent at completion of this job as a string. You can specify a Null ("") for this parameter.
EmailAddress	Specify the email address of the person you want an email sent to at the completion of this job. To send more than one email address, separate the addresses with a semicolon.
EmailWebReport	Specify whether to attach the web report to the email sent at the completion of this job. This parameter takes a string value: "True", attach the web report, "False", don't attach the web report. This parameter can be set only to "True" when the OutDestType property for the request is "Web".
EmailAttachLog	Specify whether to attach the log file to the email sent at the completion of this job. This parameter takes a string value: "True", attach the log file, "False", don't attach the log file.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.

Parameter	*Description*
JobSeqNo	Specify the job item's job sequence number within the process item level as a number.
ItemJobSeq	Specify the job item's sequence within its parent job as a number

Returns

None.

See Also

Chapter 32, "Process Request Classes," SetItemFolder, page 1858; Chapter 32, "Process Request Classes," SetOutputOption, page 1863 and Chapter 32, "Process Request Classes," AddDistributionOption, page 1846

SetItemFolder

Syntax

```
SetItemFolder(PortalFolder [, JobName] [, PrcsItemLevel] [, JobSeqNo] [,
ItemJobSeq])
```

Description

Use the SetItemFolder method to associate a report folder with any job items jobset.

The folder that you specify must have already been created using the System Settings component in Process Scheduler Manager.

Parameters

Parameter	*Description*
PortalFolder	Specify the name of the report folder to use for the job item as a string. The folder that you specify must have already been created using Report Manager.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.
JobSeqNo	Specify the job item's job sequence number within the process item level as a number.
ItemJobSeq	Specify the job item's sequence within its parent job as a number

Returns

None.

Example

```
&RQST.SetItemFolder("GENERAL", "MULTIJOB", 1, 2);
```

See Also

Chapter 32, "Process Request Classes," AddDistributionOption, page 1846

SetNotifyAppMethod

Syntax

SetNotifyAppMethod(*app_class_name,app_class_method* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])

Description

Use the SetNotifyAppMethod method to invoke your own custom application class and method to handle process status notification and the information you want to send. This method triggers the delivered PRCS_STATUS_OPER service operation, which invokes the custom method.

Important! The constructor for the application class cannot require parameters.

Note. Alternatively, you can create a custom service operation and trigger that service operation using the SetNotifyService method. Your PeopleCode program should call only one of these methods: either SetNotifyAppMethod or SetNotifyService. If both methods are used, the last method called takes precedence over the former.

Parameters

Parameter	***Description***
app_class_name	Specify the fully qualified application class name to be invoked on the target system as a string.
app_class_method	Specify the name of the method to be invoked on the target system as a string.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.

Parameter	Description
JobSeqNo	Specify the job item's job sequence number within the process item level as a number.

Returns

None.

Example

The following example contains two programs. The first program generates the process request setting process status notification through the SetNotifyAppMethod method. The second program defines the application class and method that does additional processing after the status notification has been received by the subscribing system.

Setting process status notification:

```
/************************************************************************
* Construct a ProcessRequest Object. *
***********************************************************************/
&RQST = CreateProcessRequest();
&RQST.ProcessType = "Application Engine";
&RQST.Processname = "AEMINITEST";
&RQST.RunControlID = "AEMINI";
&RQST.OutDestType = "WEB";
&RQST.OutDestFormat = "PDF";
&RQST.NotifyTextMsgSet = 65;
&RQST.NotifyTextMsgNum = 237;
&RQST.RunDateTime = %Datetime;
&RQST.TimeZone = %ServerTimeZone;
&RQST.SetNotifyAppMethod("RECEIVE_NOTIFICATION:ProcessNotification", ⇒
"ReceiveNotification");
&RQST.AddNotifyInfo("AEMINITEST Name", "Status");
&RQST.AddNotifyInfo("AEMINITEST Descr", "Status Notify");
&RQST.AddNotifyInfo("AEMINITEST OutType", "EMAIL");
&RQST.Schedule();
&PRCSSTATUS = &RQST.Status;
&PRCSINSTANCE = &RQST.ProcessInstance;
If &PRCSSTATUS = 0 Then
   MessageBox(%MsgStyle_OK, "", 65, 366, "Process Instance", ⇒
"AEMINITEST", &PRCSINSTANCE);
Else
   MessageBox(%MsgStyle_OK, "", 65, 0, "Process Instance", ⇒
"Process Not submitted");
End-If;
```

Application class and method definition:


```
class ProcessNotification
   method ProcessNotification();
   method ReceiveNotification(&_MSG As Message);
end-class;

/* Class constructor can have no parameters */
method ProcessNotification
end-method;

method ReceiveNotification
   /+ &_MSG as Message +/
   Local Rowset &rs_msg, &NotifyInfo;
   Local Message &message;
   Local string &sName, &sValue, &sName2, &sValue2;
   &rs_msg = &_MSG.GetRowset();
   /**************************************************************/
   /* Add logic you want to execute upon receiving notification */
   /* For example :                                             */
   /*  &RQST.SetNotifyAppMethod("RECEIVE_NOTIFICATION:          */
   /*  ProcessNotification", "ReceiveNotification");            */
   /*  &RQST.AddNotifyInfo("SQR Report", "XRFMENU");            */
   If &rs_msg(1).PRCS_STATUS.RUNSTATUS.Value = "9" Then
      /* process ran to success */

      &NotifyInfo = &rs_msg.GetRow(1).GetRowset(Scroll.PRCSNOTIFYATTR);

      /* if you have more name-value pairs                           */
      /* add code to traverse the rows from the PRCSNOTIFYATTR rowset*/

      /* e.g. Get the first name-value pair */
      &sName = &NotifyInfo(1).PRCSNOTIFYATTR.PRCS_ATTRIBUT_NAME.Value;
      &sValue = &NotifyInfo(1).PRCSNOTIFYATTR.PRCS_ATTRIBUT_VALU.Value;

      /* e.g. submit a process request */
      Local number &PrcsInstance;
      Local ProcessRequest &RQST;
      &RQST = CreateProcessRequest();
      &RQST.ProcessType = &sName;
      &RQST.ProcessName = &sValue;
      &RQST.RunControlID = "test";
      &RQST.RunLocation = "PSNT";
      &RQST.OutDestType = "WEB";
      &RQST.OutDestFormat = "PDF";
      &RQST.RunDateTime = %Datetime;
      &RQST.TimeZone = %ServerTimeZone;

      &RQST.Schedule();

   Else
      /* other processing */
   End-If;
   /**************************************************************/

end-method
```

See Also

Chapter 32, "Process Request Classes," AddNotifyInfo, page 1847 and Chapter 32, "Process Request Classes," SetNotifyService, page 1862

Chapter 6, "Application Classes," page 189

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Submitting and Scheduling Process Requests," Using Process Status Notifications

SetNotifyService

Syntax

```
SetNotifyService(srvc_op_name[, JobName] [, PrcsItemLevel] [, JobSeqNo])
```

Description

Use the SetNotifyService method to invoke your own custom service operation to handle process status notification and the information you want to send. Your custom service operation must use the PRCS_STATUS_MSG message definition.

Note. Alternatively, you can create a custom application class and method and invoke that method using the SetNotifyAppMethod method. Your PeopleCode program should call only one of these methods: either SetNotifyAppMethod or SetNotifyService. If both methods are used, the last method called takes precedence over the former.

Parameters

Parameter	***Description***
srvc_op_name	Specify the name of the custom service operation as a string.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.
JobSeqNo	Specify the job item's job sequence number within the process item level as a number.

Returns

None.

Example

```
/*********************************************************************
* Construct a ProcessRequest Object. *
*********************************************************************/
&RQST = CreateProcessRequest();
&RQST.ProcessType = "Application Engine";
&RQST.Processname = "AEMINITEST";
&RQST.RunControlID = "AEMINI";
&RQST.OutDestType = "WEB";
&RQST.OutDestFormat = "PDF";
&RQST.NotifyTextMsgSet = 65;
&RQST.NotifyTextMsgNum = 237;
&RQST.RunDateTime = %Datetime;
&RQST.TimeZone = %ServerTimeZone;
&RQST.SetNotifyService("PRCS_STATUS_OPER_TEST");
&RQST.AddNotifyInfo("AEMINITEST Name", "Status");
&RQST.AddNotifyInfo("AEMINITEST Descr", "Status Notify");
&RQST.AddNotifyInfo("AEMINITEST OutType", "EMAIL");
&RQST.Schedule();
&PRCSSTATUS = &RQST.Status;
&PRCSINSTANCE = &RQST.ProcessInstance;
If &PRCSSTATUS = 0 Then
   MessageBox(%MsgStyle_OK, "", 65, 366, "Process Instance", ⇒
"AEMINITEST", &PRCSINSTANCE);
Else
   MessageBox(%MsgStyle_OK, "", 65, 0, "Process Instance", ⇒
"Process Not submitted");
End-If;
```

See Also

Chapter 32, "Process Request Classes," AddNotifyInfo, page 1847 and Chapter 32, "Process Request Classes," SetNotifyAppMethod, page 1859

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Managing Service Operation Handlers," Implementing Handlers Using Application Classes and *PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker*, "Managing Service Operations," Adding Service Operation Definitions

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Submitting and Scheduling Process Requests," Using Process Status Notifications

SetOutputOption

Syntax

```
SetOutputOption(OutputType,OutputFormat,OutputDest [, JobName] [, PrcsItemLevel]
 [, JobSeqNo] [, ItemJobSeq])
```

Description

Use the SetOutputOption method to modify the output option of one or more job items in the main job.

Note. This method can also be used for a single process.

The job sequence number and job level are optional. They are required only when the same process name is referenced more than once in a job and the user intends to modify the output option of only one item.

If the ProcessRequest object contains an invalid output type for a process, you won't be able to successfully schedule a process or job.

The values for *OutputType,OutDestFormat,* and *OutputDestination* are dependent upon each other as well as on other values.

See Chapter 32, "Process Request Classes," Values for Output Type and Format, page 1841.

Parameters

Parameter	*Description*
OutputType	Specify the output type as a string.
OutputFormat	Specify the output format as a string.
OutputDestination	Specify the output destination as a string.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.
JobSeqNo	Specify the job item's job sequence number within the process item level as a number.
ItemJobSeq	Specify the job item's sequence within its parent job as a number

Returns

None.

Example

```
&RQST.SetOutputOption("Email", "HTM", "", "MULTIJOB", 1, 2);
```

See Also

Chapter 32, "Process Request Classes," OutDest, page 1869; Chapter 32, "Process Request Classes," OutDestFormat, page 1871 and Chapter 32, "Process Request Classes," OutDestType, page 1871

UpdateRunStatus

Syntax

UpdateRunStatus()

Description

Use the UpdateRunStatus method to change the RunStatus for a specific process, job, or jobset.

Parameters

None.

Returns

None.

Example

```
Local ProcessRequest &RQST;

&RQST = CreateProcessRequest();
&RQST.ProcessInstance = 5;
&RQST.RunStatus = 1; /* Cancel the request */
&RQST.UpdateRunSttus();
```

See Also

Chapter 32, "Process Request Classes," RunStatus, page 1875

ProcessRequest Class Properties

In this section, we discuss the ProcessRequest class properties. The properties are discussed in alphabetical order.

EmailAttachLog

Description

This property specifies whether or not a log file is attached to the email sent at completion of this job (or process). This property takes a Boolean value.

This property is read-write.

Example

```
&RQST.EmailAttachLog = False;   /*  Do not attach Log File */
```

EmailSubject

Description

This property specifies the text used in the subject of the email sent at completion of this job (or process). This property takes a string value.

This property is read-write.

Example

```
&RQST.EmailSubject = "SQR Report:  Cross Reference Listing";
```

EmailText

Description

This property specifies the text of the email sent at the completion of this job (or process). This property takes a string value.

This property is read-write.

Example

```
&RQST.EmailText = "This text will be displayed as the text of this email ";
```

You can also use the text from a message in the message catalog for this property.

```
&RQST.EmailText = MsgGetText(65, 117, "Sample text for email with two parameters",⇒
 &MessageParm1, &MessageParm2);
```

EmailWebReport

Description

This property specifies whether an email is sent to recipients of the report after it is posted to the report repository. The URL for the report is included in the email. This option is applicable only when the OutDesType for the request is Web. This property takes a Boolean value: True, the web report should be attached, False otherwise.

This property is read-write.

FileName

Description

This property contains the name of a file to be used with file dependant processing, that is, processes which are initiated only after a file becomes available.

For a file dependent process, the FileName property is set to override the file name specified in the process definition. You must specify the complete file path and extension for the file, as a string.

This property works with the RunLocation property. If you don't specify a run location, any value assigned to this property is ignored.

This property is read-write.

Example

The following code example initiates QE_AETESTPRG once the c:\import\ediData.dat file becomes available on PSNT server.

```
Local ProcessRequest &rqst1;

&rqst1 = CreateProcessRequest();
&rqst1.RunControlId = "2";
&rqst1.ProcessType = "Application Engine";
&rqst1.ProcessName = "QE_AETESTPRG";
&rqst1.RunLocation = "PSNT";
&rqst1.FileName="c:\import\ediData.dat";
&rqst1.Schedule();
```

See Also

Chapter 32, "Process Request Classes," RunLocation, page 1875

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Defining PeopleSoft Process Scheduler Support Information," Setting Process Definition Options

JobName

Description

This property contains the name of a job that you've defined in PeopleSoft Process Scheduler.

To schedule a job, you must assign values to JobName and RunControlID for the Schedule method to succeed.

If you're scheduling a job, you don't need to set the ProcessType property.

This property is read-write.

Example

```
&MYRQST.JobName = "3SQR";
```

See Also

Chapter 32, "Process Request Classes," RunControlID, page 1874

LanguageCd

Description

This property enables you to specify a language code for the process or job. If you don't specify a language code, PeopleSoft Process Scheduler first looks in the process run control table to retrieve a language code. If there isn't a value there, PeopleSoft Process Scheduler uses the user's language code specified in the User Definition table.

This property takes a string value.

This property is read-write.

Example

```
&MYRQST.LanguageCd = "ESP" /* Spanish */
```

NotifyTextMsgNum

Description

Use the NotifyTextMsgNum property to specify the message that should be displayed to the end user in the Event Notification popup display. This property takes a numeric value.

Note. You must specify both the message set and message number for the message to be displayed to the end user.

This property is read-write.

Example

```
&RQST.NotifyTextMsgSet = 65;
&RQST.NotifyTextMsgNum = 237;
```

See Also

Chapter 32, "Process Request Classes," NotifyTextMsgSet, page 1869

NotifyTextMsgSet

Description

Use this property to specify the message set of the text message to be displayed to the end user in the Event Notification popup display. This property takes a numeric value.

Note. You must specify both the message set and message number for the message to be displayed to the end user.

This property is read-write.

Example

```
&RQST.NotifyTextMsgSet = 65;
&RQST.NotifyTextMsgNum = 237;
```

See Also

Chapter 32, "Process Request Classes," NotifyTextMsgNum, page 1868

OutDest

Description

This property specifies the output destination for the process or job to be scheduled as a string.

Values depend on the setting for the OutDestType property:

- If OutDestType is FILE, OutDest must be the name of a directory. If the OutDest property isn't set, it defaults to the setting in the Process profile. If the Process Profile was not updated with a default destination, the Process Scheduler server that picked up the request sets the directory based on the Log/Output Directory setting found in the Process Scheduler Configuration file.
- If OutDestType is PRINTER, OutDest must be the name of a printer. If the OutDest property isn't set, it defaults to the setting in the Process profile. If the Process Profile was not updated with a default printer, the Process Scheduler server that picked up the request sets the printer based on the Default Printer setting found in the Process Scheduler Configuration file.
- If OutDestType is WEB or EMAIL, OutDest should contain the list of User IDs, Role IDs, or email addresses (for email only). If you don't set this property, it's automatically assigned to the person who submitted the request.
- If OutDestType is WEB, PeopleSoft Process Scheduler uses the OutDest property to determine who has access to the output.
- If OutDestType is EMAIL, PeopleSoft Process Scheduler uses the OutDest property to determine who to send the report output to.

  In both these cases, to identify whether it's a User ID or a Role ID, the value has to be preceded by one of the following:

  U:*Username* or **User:***Username* for a user ID

  R:*Rolename* or **Role:***Rolename* for a role ID

  If the ID is not preceded by either of these identifiers, Process Scheduler assumes it's an email address.

  Note. Each ID must be separated by a semicolon (;).

  The following are two examples:

  ```
  &RQST.OutDest = "U:PTDMO;R:Employee;robert_smith@peoplesoft.com"
  ```

  ```
  &RQST.OutDest = "User:PS;Role:Employee;sue_line@XYZ.com;simon_gree@peoplesoft.com"
  ```

- For any other value of OutDestType, this property has no effect.

You can specify directory and printer names using the UNC (Uniform Naming Convention) protocol.

This property is read-write.

Example

```
&MYRQST.OutDest = "C:\TEMP";
```

See Also

Chapter 32, "Process Request Classes," OutDestType, page 1871

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Managing PeopleSoft Process Scheduler"

OutDestFormat

Description

This property specifies the output format for the process or job to be scheduled as a string. Values depend on your settings for the ProcessType and OutDestType properties:

The values for OutDestType, OutDestFormat, and OutDest are dependent upon each other as well as on other values.

See Chapter 32, "Process Request Classes," Values for Output Type and Format, page 1841.

This property is read-write.

Example

```
&MYRQST.OutDestFormat = "RTF";
```

See Also

Chapter 32, "Process Request Classes," ProcessType, page 1873 and Chapter 32, "Process Request Classes," OutDestType, page 1871

OutDestType

Description

This property specifies the type of output for the process or job to be scheduled as a string.

If the ProcessRequest object contains an invalid output type for a process, you won't be able to able to successfully schedule a process or job.

The value specified for this property is used only if the Output Destination Type for the defined process or job defined in PeopleSoft Process Scheduler is specified as Any. Otherwise any value specified for this property is ignored.

The values for OutDestType, OutDestFormat, and OutDest are dependent upon each other as well as on other values.

See Chapter 32, "Process Request Classes," Values for Output Type and Format, page 1841.

This property is read-write.

Example

```
&MYRQST.OutDestType = "FILE";
```


See Also

Chapter 32, "Process Request Classes," ProcessType, page 1873 and Chapter 32, "Process Request Classes," RunControlID, page 1874

PortalFolder

Description

This property specifies the name of the report folder associated with a job item as a string. If you're specifying a folder, the folder that you specify must have already been created using Report Manager.

This property is read-write.

ProcessInstance

Description

This property is a system-generated identification number. PeopleSoft Process Scheduler assigns a ProcessInstance at runtime to each process or job it successfully schedules.

This property is read-write.

Example

```
&MYRQST.Schedule();
If &MYRQST.Status = 0 then
   /* process successfully scheduled */
   &ProcInst = &MYRQST.ProcessInstance;
Else
   /* do error processing */
End-If;
```

ProcessName

Description

This property specifies the name of a predefined process as a string.

To successfully schedule a process, you must assign values to ProcessName, ProcessType, and RunControlID (that is, for the Schedule method to succeed.)

This property is read-write.

Example

```
&MYRQST.ProcessName = "XRFWIN";
```

See Also

Chapter 32, "Process Request Classes," ProcessType, page 1873 and Chapter 32, "Process Request Classes," RunControlID, page 1874

ProcessType

Description

This property specifies the name of a predefined process type as a string.

To successfully schedule a process, you must assign values to ProcessName, ProcessType, and RunControlID (that is, for the Schedule method to succeed.)

The values for ProcessType depend on the types of processes you have defined in your system. There are generic process types that are delivered with your installation of PeopleSoft. These process types may include the following:

- Application Engine
- COBOL
- Crystal
- Cube
- nVision
- SQR
- WinWord
- Other

If you define your own processes, you can use the name of that process with the ProcessType property. For example, suppose you create a custom process named "Custom CBL Programs." You could use this as follows:

```
&MyRqst.ProcessType = "Custom CBL Programs";
```

Note that spaces are included in the string for ProcessType.

This property is read-write.

Example

Note that spaces are included in the string for ProcessType.


```
&MYRQST.ProcessType = "Application Engine";
```

See Also

Chapter 32, "Process Request Classes," ProcessName, page 1872 and Chapter 32, "Process Request Classes," RunControlID, page 1874

RunControlID

Description

This property returns a string that serves, along with the user ID, as a key that identifies a predefined group of parameters to be used by a process or a job at runtime.

To successfully schedule a process, you must provide values for RunControlID, ProcessName, and ProcessType.

To successfully schedule a job, you must provide values for RunControlID and JobName.

This property is read-write.

Example

```
&MYRQST.RunControlID = "MYRUNCONTROLID";
or
&MYRQST.RunControlID = PRCSSAMPLEREC.RUN_CNTL_ID;
```

See Also

Chapter 32, "Process Request Classes," JobName, page 1868; Chapter 32, "Process Request Classes," ProcessName, page 1872 and Chapter 32, "Process Request Classes," ProcessType, page 1873

RunDateTime

Description

This property contains a DateTime value that specifies when the scheduled process or job will run.

If you don't specify a value for this property, and there is no date time set for the pre-defined process or job, the process or job runs as soon as the Schedule method is executed.

This property is read-write.

Example

The following example schedules the process or job to run as soon as the Schedule method is executed:

```
&MYRQST.RunDateTime = %Datetime;
```


RunLocation

Description

This property specifies the Process Scheduler Server name the request should be scheduled on. This property takes a string value. Values for RunLocation is a specific server name, such as PSNT.

If no RunLocation is specified, the request is scheduled based on both the default operating system and load balancing options set in the System Settings page.

This property is read-write.

Example

```
&MYRQST.RunLocation = "SERVER";
or
&MYRQST.RunLocation = "PSNT";
```

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Using Process Monitor," Viewing the Status of Processes

RunRecurrence

Description

This property specifies the frequency with which a process or job is to be run as a string. The RunRecurrence value you use must be the name of a Recurrence Definition defined in Process Scheduler Manager to successfully schedule a job or process (that is, for the Schedule method to succeed.)

This property is read-write.

Example

```
&MYRQST.RunRecurrence = "M-F at 5pm";
```

RunStatus

Description

This property specifies the run status of a process request as a number. This property is used with the UpdateRunStatus method to change the status of a request. Values are:

RunStatus	*Description*
1	Cancel
2	Delete
3	Error
4	Hold
5	Queued
6	Initiated
7	Processing
8	Cancelled
9	Success
10	Not Successful
11	Posted
12	Unable to post
13	Resend
14	Posting
15	Generated
16	Pending

This property is read-write.

Example

```
&MYRQST.RunStatus = 1;
```

See Also

Chapter 32, "Process Request Classes," UpdateRunStatus, page 1865

Status

Description

This property returns a number based on the result of the last execution of the Schedule method.

Valid returns are:

- Zero if the method succeeded
- Non-zero otherwise

This property is read-only.

Example

```
&MYRQST.Schedule();
If &MYRQST.Status = 0 then
   /* Schedule succeeded. */
Else
   /* Process (job) not scheduled, do error processing */
End-If;
```

TimeZone

Description

This property contains a timezone value that specifies when the scheduled process or job will run. If no value is used for this property, the server timezone is used. This property takes a string value.

This property is read-write.

Example

```
&MyRqst.TimeZone = "EST";
```

ProcessRequest Class Examples

The following section contains the following examples:

- Scheduling a single process

- Scheduling a job where job item changes are not made
- Scheduling a job where job item changes are made

Scheduling a Single Process

The following example is when you're scheduling a single process:

```
Local ProcessRequest &RQST;

/* Create the ProcessRequest Object */
&RQST = CreateProcessRequest();

/* Set all the Required Properties */
&RQST.RunControlID = &sRunCntlId;
&RQST.ProcessType = &sProcessType;
&RQST.ProcessName = &sProcessName;

/* Set any Optional Properties for this Process */
&RQST.RunLocation = &sRunLocation;
&RQST.RunDateTime = &dttmRunDateTime;
&RQST.TimeZone = &sTimeZone;
&RQST.PortalFolder = &sPortalFolder;
&RQST.RunRecurrence = &sRecurrence;
&RQST.OutDestType = &sOutDestType;
&RQST.OutDestFormat = &sOutDestFormat;
&RQST.OutDest = &sOutputDirectory;
&RQST.EmailAttachLog = &bEmailAttachLog;
&RQST.EmailWebReport = &bEmailWebReport;
&RQST.EmailSubject = &sEmailSubject;
&RQST.EmailText = &sEmailText;

/* Schedule the Process */
&RQST.Schedule();
```

Scheduling a Job Where Job Item Changes Are Not Made

The following example is when job item changes aren't made.


```
Local ProcessRequest &RQST;

/* Create the ProcessRequest Object */
&RQST = CreateProcessRequest();

/* Set all the Required Properties */
&RQST.RunControlID = &sRunCntlId;
&RQST.JobName = &sJobName;

/* Set any Optional Main Job Properties */
&RQST.RunLocation = &sRunLocation;
&RQST.RunDateTime = &dttmRunDateTime;
&RQST.TimeZone = &sTimeZone;
&RQST.PortalFolder = &sPortalFolder;
&RQST.RunRecurrence = &sRecurrence;
&RQST.OutDestType = &sOutDestType;
&RQST.OutDestFormat = &sOutDestFormat;
&RQST.OutDest = &sOutputDirectory;
&RQST.EmailAttachLog = &bEmailAttachLog;
&RQST.EmailWebReport = &bEmailWebReport;
&RQST.EmailSubject = &sEmailSubject;
&RQST.EmailText = &sEmailText;

/* Schedule the Job */
&RQST.Schedule();
```

Scheduling a Job Where Job Item Changes Are Made

The following example is when job item changes are made.

```
Local ProcessRequest &RQST;

/* Create the ProcessRequest Object */
&RQST = CreateProcessRequest("PSJob", &sJobName);

/* Set all the Required Properties */
/* Note: the JobName Property has already */
/* been set in the call above, so you don&rsquo;t */
/* have to set it here */
&RQST.RunControlID = &sRunCntlId;

/* Set any Optional Job Item Properties */
&RQST.SetOutputOption(&sOutputType, &sOutputFormat, &sOutputDest, &sJobName, &n⇒
PrcsItemLevel, &nJobSeqNo);

&RQST.SetEmailOption(&sEmailSubject, &sEmailText, &sEmailAddress, ⇒
&sEmailWebReport, &sEmailAttachLog, &sJobName, &nPrcsItemLevel, &nJobSeqNo);
&RQST.SetItemFolder(&sPortalFolder, &sJobName, &nPrcsItemLevel, &nJobSeqNo);
&RQST.AddDistributionOption(&sDistIdType, &sDistId, &sJobName, &nPrcsItemLevel, &n⇒
JobSeqNo);

/* Set any additional Optional Main Job Properties */
&RQST.RunLocation = &sRunLocation;
&RQST.RunDateTime = &dttmRunDateTime;
&RQST.TimeZone = &sTimeZone;
&RQST.PortalFolder = &sPortalFolder;
&RQST.RunRecurrence = &sRecurrence;
&RQST.OutDestType = &sOutDestType;
&RQST.OutDestFormat = &sOutDestFormat;
&RQST.OutDest = &sOutputDirectory;
&RQST.EmailAttachLog = &bEmailAttachLog;
&RQST.EmailWebReport = &bEmailWebReport;
&RQST.EmailSubject = &sEmailSubject;
&RQST.EmailText = &sEmailText;

/* Schedule the Job */
&RQST.Schedule();
```

PrcsApi Class

Use the PrcsApi class with file dependant processing. This section includes:

- Scope of a PrcsApi object.
- Data type of PrcsApi object.
- How to import a PrcsApi object
- How to create a PrcsApi object
- PrcsApi reference

See Chapter 32, "Process Request Classes," File Dependant Processing, page 1845.

Scope of a PrcsApi Object

A PrcsApi object can be instantiated only from PeopleCode.

A PrcsApi object can be called from a component, an internet script, or an Application Engine program.

PrcsApi class objects can be of Local, Global, or Component scope.

Data Type of a PrcsApi Object

PrcsApi objects are of type PrcsApi.

```
Local PrcsApi &api = create PrcsApi();
```

How to Import the PrcsApi Class

The PrcsApi class is *not* a built-in class, like Rowset, Field, Record, and so on. It is an Application Class. Before you can use this class in your PeopleCode program, you must import it to your program.

An import statement names either all the classes in a package or one particular application class. For importing the PrcsApi class, PeopleSoft recommends that you import the API class.

The import statement you should use is as follows:

```
import PT_PRCS:API:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available.

See Also

Chapter 6, "Application Classes," page 189

How to Create a PrcsApi Object

After you've imported the PrcsApi class, you instantiate an object of that class using the constructor for the class and the *Create* function.

The following example creates a new instance of the PrcsApi class, as the variable &api, with local scope:

```
import PT_PRCS:API:*;

Local PrcsApi &api = create PrcsApi();
```


PrcsApi Class Constructor

You must use the constructor for the PrcsApi class to instantiate an instance of that class. The following is the constructor for the PrcsApi class.

PrcsApi

Syntax

`PrcsApi``()`

Description

Use the PrcsApi constructor to create an instance of the PrcsApi class.

Parameters

None.

Returns

A reference to a PrcsApi object.

PrcsApi Class Methods

In this section, we discuss the PrcsApi class methods. The methods are discussed in alphabetical order.

getAllFileNames

Syntax

`getAllFileNames`*`(PrcsInstance)`*

Description

Use the getAllFileNames method to return a list of all the files names detected by the scheduler to intiate this process. The names are returned in an array of string, as full path names.

Parameters

Parameter	*Description*
PrcsInstance	Specify the process instance for which you want to get all the associated file names, as a number.

Returns

An array of string containing all the files names associated with the specified process.

Example

The following example returns a list of file names.

```
/* QE_OPT_AET is the state record for the application engine program in this⇒
 example */

import PT_PRCS:API:*;

Local PrcsApi &api = create PrcsApi();

Local File &FileIO, &FileLog;

Local array of string &strList = &api.getAllFileNames(QE_OPT_AET.PROCESS_⇒
INSTANCE);
/* api call needs process instance as parameter */

/* the api returns list of file names matched by the scheduler for this instance */
For &i = 1 To &strList.Len
   &IOFilename = &strList [&i];

   &FileIO = GetFile(&IOFilename, "r", "a", %FilePath_Absolute);

   &FileIO.Delete();

   &FileIO.Close();
End-For;
```

notifyToWindow

Syntax

notifyToWindow*(PrcsInstance,Message)*

Description

Use the notifyToWindow method to display additional messages to the window.

If a process is not defined as going to Window, using this method has no effect. The purpose of this method is to display additional messages to the window when an Application Engine process (no other process type) is run to Window.

Parameters

Parameter	***Description***
PrcsInstance	Specify the process instance for which you want to display messages, as a number.
Message	Specify the message you want displayed, as a string.

Returns

A number: 0 if method failed.

Example

The following PeopleCode would be in a step in an application engine program.

```
import PT_PRCS:API:*;

Local PrcsApi &api = create PrcsApi();

/* QE_AESTATUS_AET is the sate record for this Application Engine program */

&nret = &api.notifyToWindow(QE_AESTATUS_AET.PROCESS_INSTANCE, "Hi There! this is⇒
 First step.");

/* this displays the message on the window */
```


Chapter 33

Query Classes

This chapter provides an overview of the query classes and discusses the following topics:

- Collections in the query classes.
- Life cycle of a query.
- Query classes hierarchy.
- Query API overview.
- Working with query criteria and expressions.
- Query monitor.
- Using query metadata.
- Running a query.
- Specifying the user's language.
- Query security.
- Error handling with query classes.
- Understanding QueryOutputFields and QuerySelectedFields.
- Understanding having criteria.
- Data type of query objects.
- Scope of query objects.
- Query classes reference.
- Query classes examples.

Understanding Query Classes

You create queries using PeopleSoft Query Manager to extract the data you need from the PeopleSoft database. You can use the query classes in PeopleCode to create a new query, or to modify or delete an existing query. You can also use methods in the Query class to execute the query and have the result set returned as either a rowset or have it format and write the result set to a file. You can also use the query classes to create a SQL statement to be used with the SQL object. In fact, the query classes expose all of the attributes and methods needed by the PeopleSoft Query Manager and Query Viewer applications.

Creating or deleting a query object does not create or delete query information. You must call the appropriate method for that query object directly to create or delete database information, that is, the Create or Delete method.

All of the classes, and most of the properties and methods that make up the query classes, have a GUI representation in PeopleSoft Query Manager. This document assumes that the reader has working knowledge of PeopleSoft Query Manager.

There aren't any external built-in functions for the query classes: objects are instantiated from other objects or from a session object.

See Also

Chapter 38, "Session Class," page 2179

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "Getting Started with PeopleSoft Query"

Collections in the Query Classes

A *collection* is a set of similar things, like a group of already existing queries or QueryRecords. As with everything else in the query classes, collections have a GUI representation in PeopleSoft Query.

For example, when you want to open a query, the search page returns a list of all the available queries. This is equivalent to using the FindQueries session class method to get a Query collection.

The following collections are part of the query classes:

- QuerySelect collection
- QueryDBRecord collection
- QueryDBRecordField collection
- Query collection
- QueryRecord collection
- QueryOutputField collection
- QuerySelectedField collection
- QueryCriteria collection
- QueryExpression collection
- QueryPrompt collection

Life Cycle of a Query

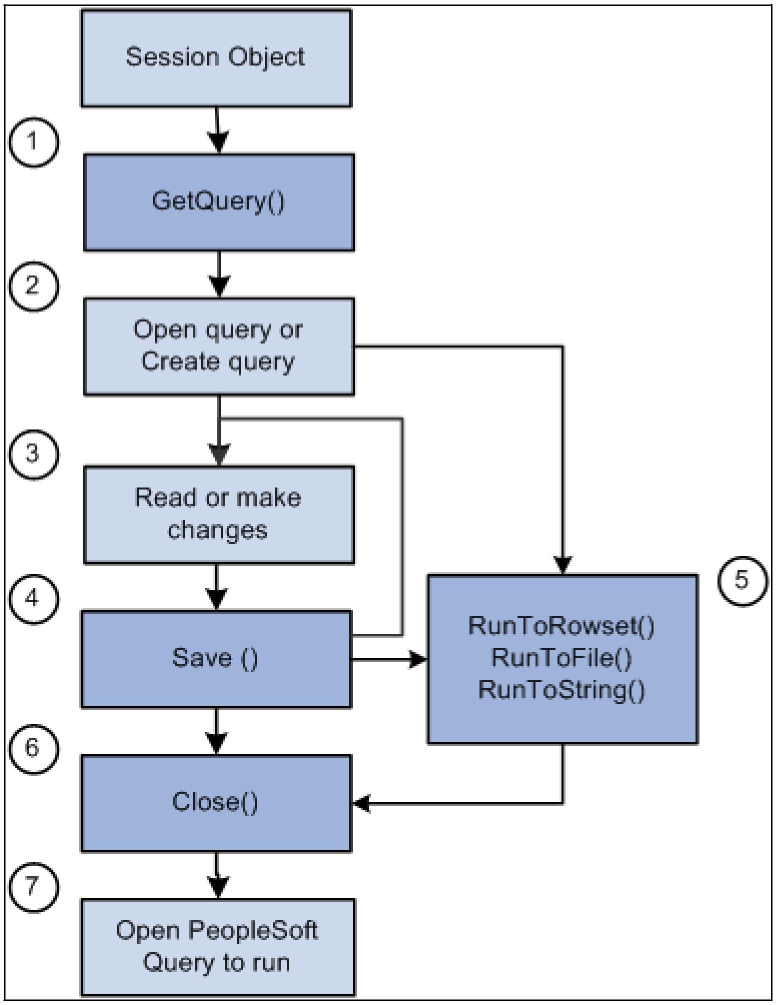
At runtime, there are certain things you want to do with a query, like creating one from scratch, updating the criteria for an existing query, running a query, and so on. The following is an overview of this process, and assumes the most common method to use the query API. These steps are expanded in other sections.

1. Invoke the GetQuery method on the PeopleSoft session object to get a query.
2. Either open the specific query you want using Open, or create a new query using Create.
3. Read the query statistics, or make changes as appropriate, adding records, field, criteria, and so on.
4. Save the changes.
5. (Optional) Use the RunToRowset method to run the query.

   Note. Be careful whenever you write a PeopleCode program that uses the RunToRowset method because this method could return a large amount of data that could potentially exceed the memory available. For this reason, RunToRowset should be used only when you know that the query being executed returns a reasonable amount of data, or be sure to use the MaxRows parameter to control the maximum amount of data that can be returned.

   Alternatively, you can:

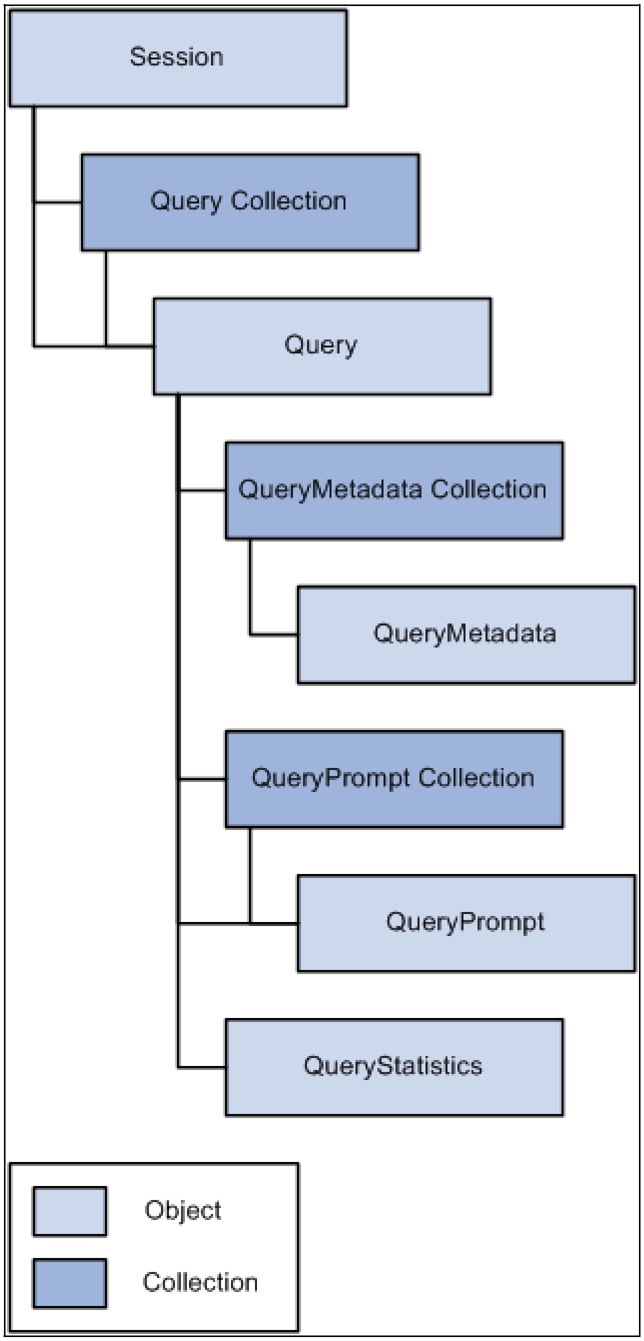
   - Use the RunToFile method to execute a scheduled query.
   - Use the RunToString method to run a query and return the result as a formatted string.
6. Close the query.
7. If you want, you can navigate to PeopleSoft Query Manager, Query Viewer or the Query Designer to run the query.



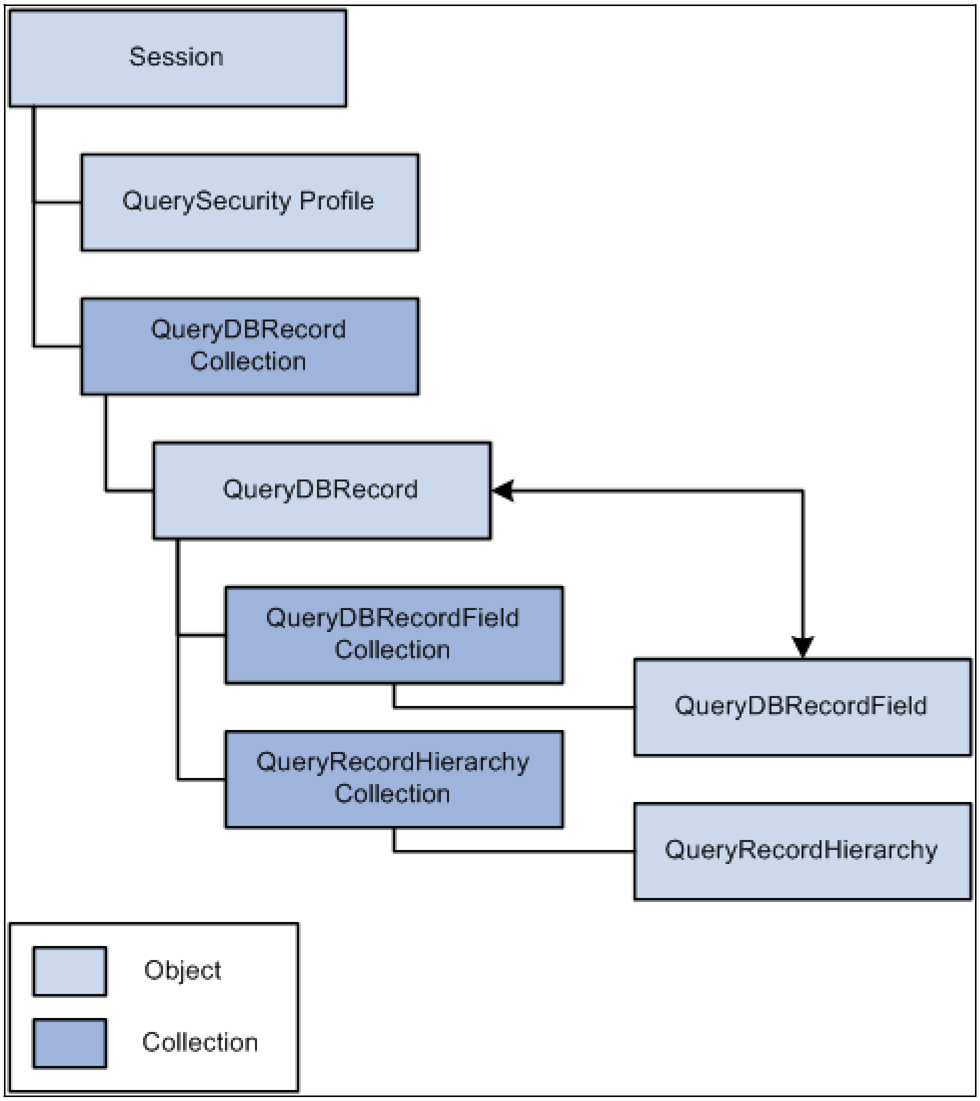
Life cycle of a Query object

Query Classes Hierarchy

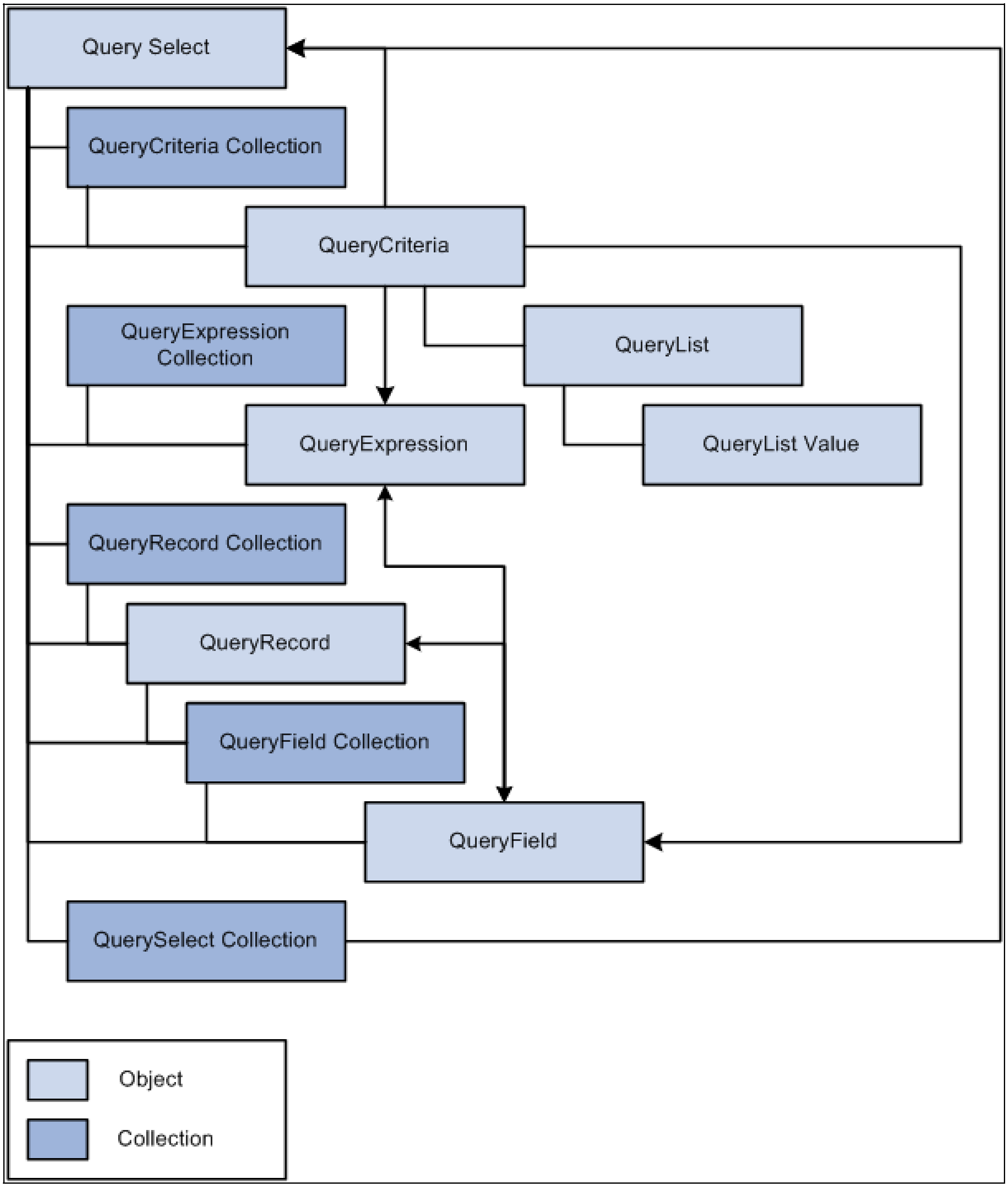
There are many different classes used with the query API. The following flowcharts illustrate all the different classes and how they're interrelated.



Query API class hierarchy (part 1 of 3)



Query API class hierarchy (part 2 of 3)



Query API class hierarchy (part 3 of 3)

Query API Overview

The query API is made up of many classes. The following are the primary parts, generally used when updating a query:

Query	The query definition.
QuerySelect	A query SELECT statement. There can be multiple QuerySelect objects for queries that involve unions or subqueries. Each select (or union or subquery) consists of QueryRecords, QueryOutputFields, QuerySelectedFields, and QueryCriteria.
QueryRecords	The records that are part of the existing QuerySelect definition.
QueryOutputFields	The fields that you've selected to be displayed when the results of the query are run.
QuerySelectedFields	All the fields that make up the QuerySelect Definition. These include the fields selected as output fields, and fields added as part of Query Expressions.
QueryCriteria	The criteria for the query.
QueryExpression	The Query expressions that can contain SQL functions or other SQL fragments.
QueryList	The lists used in the in-list functionality for criteria.
QueryPrompt	The prompts used in criteria.
QueryDBRecords and QueryDBRecordFields	All the records available to be used as QueryRecords, and all the fields available. This list is restricted based on a user's Query Access Security Groups.

Query

A database query. You must get a query before you can access any of the other query classes. You can then create a new query and save it to the database, or open an existing query and modify it.

QuerySelect

Each query is composed of one or more select statements:

Main Select	The instance of the first SELECT statement of the Query is the Main Select. There can only be one Main Select statement in the Query. This instance consists of the QueryOutputFIelds, QueryCriteria, and the QueryExpressions for the Main Select statement.
Union	In addition to the Main Select, a query can have one or more unions. The Unions are added by using the AddUnion method.
Sub-Select	A subquery used in the criteria of the Main Select or in a Union.

QueryRecords

The records that are part of an existing query definition. In PeopleSoft Query, these are the records listed on the query tab. Each QuerySelect has its own set of QueryRecords.

QueryOutputFields

The fields that you've selected to be part of the query definition. They're called output fields because when you run the query, these are the fields that make up the output columns.

In PeopleSoft Query, these are the fields listed on the Fields tab.

The collection of QueryOutputFields does *not* necessarily include all columns returned in query resultset. This collection only includes columns that have been added to query output collection using the query classes or by an end-user designing a query.

PeopleSoft Query can also add additional columns for related language processing and also for translate labels on fields, and so you cannot use the Query Output Collections as a way to discover all of the columns that are returned in the resultset or by executing the SQL generated from a query.

QuerySelectedFields

All the fields that make up the query definition. These include the fields selected as output fields, and fields added as QueryExpressions.

QueryCriteria

The selection criteria for the query. Each QueryCriteria object is made up of the following:

Logical	Any criteria objects after the first must include have a Logical value, either AND or OR.
Expression 1	A field or value that you want to base the selection criterion on, that is, Expression1 is the left-hand side of the criterion's comparison.
Operator	A mathematical or other operator used to specify the relationship between Expression 1 and Expression 2.
Expression 2	A field or other value, also called a comparison value, used with Expression 1, that is, Expression2 is the right-hand side of the criterion's comparison.

In PeopleSoft Query, a QueryCriteria is under the Criteria tab.

Expression can be made up of constant values, fields, subqueries, and so on.

See Also

Chapter 33, "Query Classes," Working With Query Criteria and Expressions, page 1894

QueryDBRecords and QueryDBRecordFields

A QueryDBRecord is a record in the database that can be used as a QueryRecord. The list of records is controlled by security: the only records displayed as QueryDBRecords are records accessible by the user.

The QueryDBRecordFields are the fields that make up the QueryDBRecords.

In PeopleSoft Query, the QueryDBRecords are under the Record tab. After you click on the plus sign next to a record, the QueryDBRecordFields are displayed.

Working With Query Criteria and Expressions

If you run a query after selecting the QueryFields (which executes a SQL statement, such as SELECT EMPLID, DEPTID from PS_QE_EMPLOYEE), the system retrieves *all* the data in those columns; that is, it retrieves the data from every row in the QueryRecord or records because there is no filter limiting the number of rows.

You can select which *rows*of data you want by adding selection criteria to the query.

This document assumes that you know how to use selection criteria. This section discusses working with the QueryCriteria and QueryExpression objects in the query API only.

This section discusses how to:

- Set the expression type.
- Add new expressions.
- Add an operator and Expression 2 dependencies.
- Set a drilling URL.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "Defining Selection Criteria"

Setting the Expression Type

Before you can add a new expression (either Expression 1 or Expression 2) to your QueryCriteria, you must set the type for the expression.

The following code example adds a new criteria, sets the type for the first expression, adds the first expression to the main select of the query definition, then does the same thing for the second expression.


```
&MyQuerySelect = &MyQuery.QuerySelect;
&MyCriteria = &MyQuerySelect.AddCriteria();

/* make expression 1 a field */
&MyCriteria.Expr1Type = %Query_ExprField;

/* add the ABSENCE_TYPE field */
&MyCriteria.AddExpr1Field("A", "ABSENCE_TYPE");

/* make it not equal to */
&MyCriteria.Operator = %Query_CondNotEqual;

/* Make expression 2 a constant */
&MyCriteria.Expr2Type = %Query_ExprConstant;
&MyCriteriaExpr = &MyCriteria.AddExpr2Expression1();
&MyCriteriaExpr.Text = "VAC";
```

Adding New Expressions

When you use any of the QueryCriteria methods to add either a new Expression 1 or Expression 2, you destroy the existing value.

In general, you should use the QueryCriteria methods to add a new Expression 1 or Expression 2 only when you're adding a new criteria to a query.

Adding an Operator and Expression 2 Dependencies

Which values are valid for the Expression 2 type (Expr2Type property) depend on the value of the Operator property.

The following table describes which Expr2Type values are valid with which values of Operator.

Operator	***Expression 2***
equal to not equal to greater than not greater than less than not less than	Constant Field Expression Subquery Prompt
Exists not exists	Subquery
Like not like	Constant (with wildcards) Prompt

Operator	*Expression 2*
is null is not null	
in tree not in tree	Tree Option Tree Prompt Option
Eff Date <= Eff Date >= Eff Date < EffDate >	Field Expression Constant Current Date
First Eff Date Last Eff Date	
in list not in list	In list Subquery
Between Not Between	Const-Const Const-Field Const-Expr Field-Const Field-Field Field-Expr Expr-Const Expr-Field Expr-Expr

See Also

Chapter 33, "Query Classes," Operator, page 2012

Setting a Drilling URL

A *drilling* URL provides a clickable link in query results allowing a user to drill from the query results to another query, to another component, or to an external site. A drilling URL is defined as a special type of query expression. Like other query expressions, the user can set the drilling URL on the Edit Expression Properties page, or the drilling URL can be set in a PeopleCode program as follows:


```
/* Setting a drilling URL */

&aQueryExpr = &aQuerySelect.AddExpression(&sExprName);

/* Set the expression type to drilling URL */
&aQueryExpr.Type = %FieldType_URL;

/* Set the expression text and number from record fields */
/* The Text property must conform to expected formats    */
&aQueryExpr.Text = &rRecordExpr.QRYCRIT1EXPRTEXT.Value;
&aQueryExpr.ExpNum = &rRecordExpr.QRYCRIT1EXPRNUM.Value;
```

For a drilling URL, the Text property of the QueryExpression object must conform to one of three expected formats. The value of the Text property is not validated; therefore, it is the calling program's responsibility to ensure that value is complete and correctly formatted. Each of the drilling URL types has a different format as follows:

- The drilling URL for a component requires the following format:

  '/c/*menu.component.market***?Action=U***search_key*=*unique_field***:**⇒
  *unique_field_URL_is_mapped_to***'**

  For example:

  ```
  '/c/QE_SAMPLE_APPS.QE_DEPT_TBL.GBL?Action=U&DEPTID=A.DEPTID&SETID=A.SETID:A.SETID'
  ```

- The drilling URL for a query requires the following format:

  '/q/?ICAction=ICQryNameURL={**PUBLIC**|**PRIVATE**}**.**query_name**&***unique_prompt_key***=**⇒
  unique_field:*unique_field_URL_is_mapped_to***'**

  For example:

  ```
  '/q/?ICAction=ICQryNameURL=PUBLIC.DESTINATION&BIND1=A.DEPTID:B.DEPTID'
  ```

- The drilling URL for an external site requires the following format:

  '/e/?url=[*full_external_URL***]:***unique_field_URL_is_mapped_to***'**

  For example:

  ```
  '/e/?url=[http://www.yahoo.com]:A.SETID'
  ```

See Also

Chapter 33, "Query Classes," AddTrackingURL, page 1924

Chapter 33, "Query Classes," SetTrackingURL, page 1944

Chapter 33, "Query Classes," Text, page 2021

Chapter 33, "Query Classes," Type, page 2022

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "Defining Selection Criteria," Drilling URL in Oracle PeopleSoft Query

Query Monitor

The query classes provide many methods and properties for examining the QueryStatistics, which you can use to report on the average execution time, the last date and time the query was run, and so on.

You can view the statistics for a query before you save it to the database.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "Getting Started with PeopleSoft Query"

Using Query Metadata

There are two ways of accessing information about a query:

- The query classes (QueryOutputField, QueryRecord, and so on).
- The Metadata property.

If you use the Metadata property, the information is presented in a different format. It is also read-only. Using this property can be a quick and easy way to access information about a query.

Each Query Metadata object is a name-value pair. *Name* is an indicator of which Query Metadata property you're accessing, and *Value* is the value of that property.

While the value of each Query Metadata property is unique, the name may or may not be. For example, there is only one description (Descr) for each query, so there is only one Query Metadata property with *name* equal to Descr and *value* equal to the description for the query.

However, there may be more than one record for a query. A Query Metadata property exists for each record, with *name* equal to Record and *value* equal to one of the records in the query. The same is true for Input Param, Expression, Field, and Heading.

The following is a simple PeopleCode program to get the Query Metadata for a private query, ADDRESS_TEMP:


```
Local ApiObject &MyQuery, &MyMetacol, &MyMetadata;
Local File &MyFile;

&MyFile = GetFile("Metadata.Txt", "A");

&MyQuery = %Session.GetQuery();

&Rlst = &MyQuery.Open("ADDRESS_TEMP", False, 1);

If &Rlst = 0 Then
   &MyMetacol = &MyQuery.metadata;
   &MyFile.WriteLine("Name              Value");
   &MyFile.WriteLine("-----------------------");

   For &i = 1 To &MyMetacol.count
      &MyMetadata = &MyMetacol.item(&i);
      &Name = &MyMetadata.name;
      &Value = &MyMetadata.Value;
      &MyFile.WriteLine(&Name | "              " | &Value);
   End-For;

Else
   WinMessage("Open query not successful");
End-If;
```

Here is the sample output:

```
Name                  Value
-----------------------
Descr                 Temporary address query for testing
LongDescr
Public/Private        QEDMO
LastUpdDttm           2001-11-19-15.06.22.930000
LastUpdOprId          QEDMO
Record                QE_PERS_DATA
Field                 QE_EMPLID
Field                 QE_NAME
Field                 ADDRESS1
Field                 ADDRESS2
Field                 ADDRESS3
Field                 ADDRESS4
Field                 CITY
Field                 COUNTY
Field                 STATE
Field                 QE_ZIP
Field                 QE_COUNTRY
Heading               ID
Heading               Name
Heading               Address 1
Heading               Address 2
Heading               Address 3
Heading               Address 4
Heading               City
Heading               County
Heading               St
Heading               QE_ZIP
Heading               QE_COUNTRY
```

Running a Query

With query API, you have the following options for running a query:

- Using the RunToRowset method.

  This method takes a QueryPrompt record as input and returns a PeopleCode rowset containing the query result. You should always use a standalone rowset (that is, one that was created using CreateRowset).

- Using the RunToFile method.

  This method takes a QueryPrompt record and a file name as input and writes the query result to the file.

- Using the RunToString method.

  This method takes a QueryPrompt record and writes the query result to a formatted string.

Note. For the query classes, all date, time, and datetime fields that are part of a query are now required fields when running a query.

Considerations Using the RunTo Methods

RunToRowset may require a lot of memory for the Rowset object; therefore, it also takes more processing time. PeopleSoft recommends that RunToRowset not be used for retrieving large result sets, on the order of 50,000 or more items.

All of the RunTo methods can be called to run a query before saving it—that is, it isn't necessary to first save the query.

The last parameter of all of the RunTo methods is the maximum number of rows to fetch.

- -1 returns all rows regardless of the setting on the security profile.
- 0 returns the maximum number of rows allowed by the security profile.
- >0 is the limit on the number of rows.

See Also

Chapter 33, "Query Classes," RunToFile, page 1935

Chapter 33, "Query Classes," RunToRowset, page 1938

Chapter 33, "Query Classes," RunToString, page 1941

Specifying the User's Language

For scheduled queries, the system uses the language specified in the user's profile. It does not use the language selected during signon. The system also uses the International and Regional settings the user specified using My Personalizations. If no personal setting have been specified, the system uses the default installation international settings.

Note. Most PeopleSoft components can default to international settings from the browser if the user has not set any user specific settings. However, this is not available for scheduled queries or any Process Scheduler processes.

Query Security

Security is critical for your business data. Typically, you don't want everyone in your company to have access to all the data. All the standard security features used with PeopleSoft applications are integrated in the PeopleSoft Query, as well as the query classes.

In addition, you can use the QuerySecurityProfile Class to view the current user's security profile for PeopleSoft Query. This class doesn't contain any methods, and all the properties are read-only.

See Also

Chapter 33, "Query Classes," QuerySecurityProfile Class, page 2037

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "PeopleSoft Query Security"

PeopleTools 8.51 PeopleBook: Security Administration, "Understanding PeopleSoft Security"

Error Handling With Query Classes

All errors for the query classes, like the other APIs, are logged in the PSMessages collection, instantiated from a session object. In addition, some methods return error codes. See the individual method description to see if it returns anything.

The query classes log errors "interactively", that is, as they happen. For example, suppose you specified an invalid query name. The error would be logged in the PSMessages collection as soon as you executed the GetQuery method.

When to check for errors is application-specific. However, if you check for errors after every assignment, you may see a performance degradation.

PeopleSoft recommends that you check for invalid prompts after using any of the following methods or properties:

- Save, RunToRowset, or RunToFile Query class methods.
- SQL, RuntimePrompts, or Prompts Query class properties.

- Any of the methods or properties in the QueryPrompts collection.

The easiest way to check for errors is to check the number of messages in the PSMessages collection, using the Count property. If the Count is 0, there are no errors.

```
Local ApiObject &MySession;
Local ApiObject &ERRORCOL;
Local ApiObject &Query, &QueryList;

&MySession = %Session;

If &MySession Then
   /* connection is good */

   &QueryList = &MySession.SearchPublicQueries(%Query_ListQuery, ⇒
%Query_FindName, "%", True);

   For &I = 1 to &QueryList.Count
      &Query = &QueryList.Item(&I);

      /* Do processing */

      /* Do error checking */

      &ERRORCOL = &MySession.PSMessages;
      If (&ERRORCOL.Count <> 0) Then
         /* errors occurred - do processing */
      Else
         /* no errors */
      End-If;

   End-For;
Else
   /* do processing for no connection */
End-If;
```

See Also

Chapter 38, "Session Class," Error Handling, page 2180

Understanding QueryOutputFields and QuerySelectedFields

If you select a field from a Query Record, it becomes a QueryOutputField, that is, it becomes one of the columns in the SQL output. QuerySelectedFields consist of displayed fields and Query Expression Fields.

QuerySelectedFields and QueryOutputFields have all the same methods and properties, so in this documentation, they're described together under the heading QueryField.

The collection of QueryOutputFields does *not* necessarily include all columns returned in query resultset. This collection only includes columns that have been added to query output collection using the query classes or by an end-user designing a query.

PeopleSoft Query can also add additional columns for related language processing and also for translate labels on fields, and so you cannot use the Query Output Collections as a way to discover all of the columns that are returned in the resultset or by executing the SQL generated from a query.

Understanding Having Criteria

You can have where (simple) criteria or having criteria in a query. In most cases, you have a WHERE clause only—that is, simple criteria. Having criteria is only used in some special cases of aggregation queries. They represent the HAVING clause of the SQL statement, similar to the following:

```
select A.DEPTID, COUNT(A.EMPLID)

from PS_QE_EMPLOYEE A

group by A.DEPTID

having COUNT(A.EMPLID)>5
```

Having criteria are separated from simple criteria. They're accessed as follows:

- Having criteria are accessed using either the AddHavingCriteria QuerySelect method or HavingCriteria QuerySelect property
- Simple criteria are accessed using either the AddCriteria QuerySelect method or the Criteria QuerySelect property.
- However, the having criteria and the simple criteria have all the same methods and properties, so in this documentation, they're described together under the heading QueryCriteria.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "Defining Selection Criteria," Defining HAVING Criteria

Data Type of Query Objects

All query objects, like a query collection, a QueryDBRecord, a QueryExpression, and so on, are declared as type ApiObject. For example,

```
Global ApiObject &MyQuery;

Local ApiObject &MyExpression;
```

Scope of Query Objects

All query objects can be instantiated from PeopleCode only.

You can use this object anywhere you have PeopleCode—that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

You can only instantiate a query object from a Session object. You *must* instantiate the Session object before you can instantiate a query object or query collection. Use the %Session system variable to connect to the existing session.

```
Local ApiObject &QueryList;
Local ApiObject &MySession;

&MySession = %Session;

If &MySession <> Null Then
   /* connection is good */
   /* Search for public queries of type QUERY, search both the name and*/
   /* description for the pattern MyQ%, and do a case insensitive search */

   &QueryList = &MySession.SearchPublicQueries(1, 3, "MyQ%", False);
Else
   /* do error processing */
End-if;
```

Query Classes Reference

This section provides a reference to the following topics:

- Session class methods in the query API.
- Query collection.
- Query class.
- QuerySelect collection.
- QuerySelect class.
- QueryRecord collection.
- QueryRecord class.
- QueryField collection.
- QueryField class.
- QueryCriteria collection.
- QueryCriteria class.
- QueryExpression collection.
- QueryExpression class.
- QueryList class.
- QueryListValue class.
- QueryRecordHierarchy collection.
- QueryRecordHierarchy class.

- Query Metadata collection.
- Query Metadata class.
- QueryStatistics class.
- QuerySecurityProfile class.
- QueryDBRecord collection.
- QueryDBRecord class.
- QueryDBRecordField collection.
- QueryDBRecordField class.
- QueryPrompt collection.
- QueryPrompt class.

Session Class Methods in the Query API

The following methods are part of the query API. However, they're used with a Session object.

See Also

Chapter 38, "Session Class," page 2179

AdvancedSearchQueries

Syntax

```
AdvancedSearchQueries(GetFavorites,QueryName,QueryNameOp,Descr,DescrOp,
FolderName,FolderNameOp, RecordName,RecordNameOp,FieldName,FieldNameOp,TreeName,
TreeNameOp,QueryType,OwnerType,CaseSensitive)
```

Description

Use the AdvancedSearchQueries method to do more complex searches for queries.

Using Search Operators

All of the parameters for this method work in pairs, with the value in the first of the paired parameters further distinguished by the search operator used in the second parameter. For example, the value specified in *FieldName* is paired with the value specified in *FieldNameOp.*

All of the search operator parameters use the following values. Note that the format of the value of the first parameter is sometimes affected by the value of the search operator parameter.

Constant	*Description*
%Query_AdvSrchBegins	Name begins with the values specified.
%Query_AdvSrchContains	Name contains the value specified.
%Query_AdvSrchEquals	Name equals the value specified.
%Query_AdvSrchNotEquals	The name does not equal the value specified.
%Query_AdvSrchLessThan	The name is less than the value specified.
%Query_AdvSrchLessEquals	The name is less than or equal to the value specified.
%Query_AdvSrchGreaterThan	The name is greater than the value specified.
%Query_AdvSrchGreaterEquals	The name is greater than or equal to the value specified.
%Query_AdvSrchBetween	The name is between two values specified by a comma. Do not use quotation marks. For example, ACCT1,ACCT9.
%Query_AdvSrchIn	The name is in the list specified list. The values are separated with commas. Do not use quotation marks. For example, ACCT1, ACCT2, ACCT3

Parameters

Parameter	*Description*
GetFavorites	Specify whether to return only queries marked as favorites. This parameter takes a Boolean value: true if you only want favorite queries returned, false otherwise. If you specify true for this parameter, all other parameters are ignored.
QueryName	Specify the name of the query you want returned, as a string. Use this parameter with the *QueryNameOp* parameter.
QueryNameOp	Specify the operator to be used with the *QueryName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
Descr	Specify the description you want returned, as a string. Use this parameter with the *DescrOp* parameter.
DescrOp	Specify the operator to be used with the *Descr* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
FolderName	Specify the name of the folder of the query or queries you want returned, as a string. Use this parameter with the *FolderNameOp* parameter.
FolderNameOp	Specify the operator to be used with the *FolderName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.

Parameter	***Description***
RecordName	Specify the name of the record used with the query (queries) you want returned, as a string. Use this parameter with the *RecordNameOp* parameter.
RecordNameOp	Specify the operator to be used with the *RecordName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
FieldName	Specify the name of the field associated with the query (queries) you want returned, as a string. Use this parameter with the *FieldNameOp* parameter.
FieldNameOp	Specify the operator to be used with the *FieldName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
TreeName	Specify the name of the tree associated with the query (queries) you want returned, as a string. Use this parameter with the *TreeNameOp* parameter.
TreeNameOp	Specify the operator to be used with the *TreeName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
QueryType	Specify the type of query, as a string. See below.
OwnerType	Specify the type of owner, whether the query is public or private.
CaseSensitive	This parameter has not yet been implemented.

The values for *QueryType* can be as follows:

Numeric Value	***Constant Value***	***Description***
1	%Query_Query	Find queries of the type Query.
5	%Query_DBAgent	Find queries of the type Process.
4	%Query_Role	Find queries of the type Role
7	%Query_Archive	Find queries of the type Archive.

Returns

A reference to a Query collection containing zero or more queries.

Note. If the result set contains more than 300 rows, only the first 300 rows are returned.

AdvancedSearchRecords

Syntax

`AdvancedSearchRecords``(`*`RecordName, RecordNameOp, Descr, DescrOp, FieldName, FieldNameOp, TreeName, TreeNameOp, CaseSensitive`*`)`

Description

Use the AdvancedSearchRecords method to do more complex searches for records.

Security applies to the results of this list, that is, you have access to all the records your user ID (permission list) allows you to access.

Using Search Operators

All of the parameters for this method work in pairs, with the value in the first of the paired parameters further distinguished by the search operator used in the second parameter. For example, the value specified in *FieldName* is paired with the value specified in *FieldNameOp.*

All of the search operator paramerters use the following values. Note that the format of the value of the first parameter is sometimes affected by the value of the search operator parameter.

Constant	*Description*
%Query_AdvSrchBegins	Name begins with the values specified.
%Query_AdvSrchContains	Name contains the value specified.
%Query_AdvSrchEquals	Name equals the value specified.
%Query_AdvSrchNotEquals	The name does not equal the value specified.
%Query_AdvSrchLessThan	The name is less than the value specified.
%Query_AdvSrchLessEquals	The name is less than or equal to the value specified.
%Query_AdvSrchGreaterThan	The name is greater than the value specified.
%Query_AdvSrchGreaterEquals	The name is greater than or equal to the value specified.
%Query_AdvSrchBetween	The name is between two values specified by a comma. Do not use quotation marks. For example, ACCT1,ACCT9.
%Query_AdvSrchIn	The name is in the list specified list. The values are separated with commas. Do not use quotation marks. For example, ACCT1, ACCT2, ACCT3

Parameters

Parameter	***Description***
RecordName	Specify the name of the record you want returned, as a string. Use this parameter with the *RecordNameOp* parameter.
RecordNameOp	Specify the operator to be used with the *RecordName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
Descr	Specify the description you want returned, as a string. Use this parameter with the *DescrOp* parameter.
DescrOp	Specify the operator to be used with the *Descr* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
FolderName	Specify the name of the folder of the records you want returned, as a string. Use this parameter with the *FolderNameOp* parameter.
FolderNameOp	Specify the operator to be used with the *FolderName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
FieldName	Specify the name of the field associated with the record you want returned, as a string. Use this parameter with the *FieldNameOp* parameter.
FieldNameOp	Specify the operator to be used with the *FieldName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
TreeName	Specify the name of the tree associated with the record you want returned, as a string. Use this parameter with the *TreeNameOp* parameter.
TreeNameOp	Specify the operator to be used with the *TreeName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
CaseSensitive	**Note.** This parameter has not been implemented yet. Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it isn't.

Returns

A reference to a QueryDBRecord collection containing zero or more records.

Note. If the result set contains more than 300 rows, only the first 300 rows are returned.

FindQueryDBRecords

Syntax

`FindQueryDBRecords``()`

Description

The FindQueryDBRecords method returns a reference to a QueryDBRecord collection, filled with zero or more records.

Security applies to the results of this list, that is, you have access to all the records your user ID (permission list) allows you to access.

Parameters

None.

Returns

A reference to a QueryDBRecord collection containing zero or more records.

See Also

Chapter 33, "Query Classes," QueryDBRecord Collection, page 2042 and Chapter 33, "Query Classes," QueryDBRecord Class, page 2045

FindQueries

Syntax

`FindQueries``()`

Description

The FindQueries method returns a reference to a Query collection, filled with zero or more queries.

FindQueries Considerations

FindQueries returns both public and private queries. It depends on your database whether the private query is returned first or the public query. If you have two queries with the same name, it depends on your database whether the first use of Item returns the private or the public query.

Parameters

None.

Returns

A reference to a Query collection containing zero or more queries.

Example

In the following example, all available queries are returned:

```
Local ApiObject &MySession;
Local ApiObject &MyList;

&MySession = %Session
&MyList = &MySession.FindQueries();
```

See Also

Chapter 33, "Query Classes," Open, page 1933; Chapter 33, "Query Classes," FindQueriesDateRange, page 1911 and Chapter 33, "Query Classes," Query Collection, page 1918

FindQueriesDateRange

Syntax

FindQueriesDateRange*(StartDateString,EndDateString)*

Description

The FindQueriesDateRange method returns a reference to a Query collection, filled with zero or more queries that match the specified date range.

FindQueriesDateRange Considerations

FindQueriesDateRange returns both public and private queries. It depends on your database whether the private query is returned first or the public query. If you have two queries with the same name, it depends on your database whether the first use of Item returns the private or the public query.

Parameters

Parameter	*Description*
StartDateString	Specify the year, month, and day of the beginning date that you want to look for. This parameter takes a string value. You can specify the date either as YYYY-MM-DD *or* YYYY/MM/DD.
EndDateString	Specify the year, month, and day of the end date. This parameter takes a string value. You can specify the date either as YYYY-MM-DD *or*YYYY/MM/DD.

Returns

A reference to a Query collection containing zero or more queries.

Example

```
Local ApiObject &MySession, &QueryList;

&MySession = %Session;

&Start = GetField(VOLUN_ACT_WRK.START_DT_STR).Value;
&End = GetField(VOLUN_ACT_WRK.END_DT_STR).Value;

&QueryList = &MySession.FindQueriesDateRange(&Start, &End);
```

See Also

Chapter 33, "Query Classes," FindQueries, page 1910; Chapter 33, "Query Classes," Open, page 1933 and Chapter 33, "Query Classes," Query Collection, page 1918

GetQuery

Syntax

GetQuery()

Description

The GetQuery method returns an empty query object. After you have an empty query object, you can use it to open an existing query (using the Open method) or to create a new query definition (using the Create method).

Parameters

None.

Returns

A reference to an empty query object if successful, NULL otherwise.

Example

```
&MyQuery = &MySession.GetQuery();
```

```
If &MyQuery.Open("PHONELIST") Then
```

See Also

Chapter 33, "Query Classes," FindQueries, page 1910; Chapter 33, "Query Classes," FindQueriesDateRange, page 1911; Chapter 33, "Query Classes," Open, page 1933 and Chapter 33, "Query Classes," Create, page 1926

GetQuerySecurityProfile

Syntax

`GetQuerySecurityProfile``()`

Description

Use GetQuerySecurityProfile to return the current user's security profile for PeopleSoft Query. You can then use the QuerySecurityProfile properties to determine if the user can modify queries, the maximum number of rows to fetch for this user, and so on.

Parameters

None.

Returns

A reference to a QuerySecurityProfile if successful, NULL otherwise.

Example

```
&MySecProfile = %Session.GetQuerySecurityProfile();
If &MySecProfile.CanModifyQuery Then
   /* do some processing */
End-If;
```

See Also

Chapter 33, "Query Classes," QuerySecurityProfile Class, page 2037

SearchQueryDBRecords

Syntax

SearchQueryDBRecords*(SearchType,Pattern,CaseSensitive)*

Description

The SearchQueryDBRecords method returns a reference to a QueryDBRecord collection, filled with zero or more records.

Security applies to the results of this list, that is, you have access to all the records your user ID (permission list) allows you to access.

You can use wildcard characters % and _ when searching. % means find all characters, while _ means find a single character. For example, if you wanted to find all queries that started with the letter M, use "M%" for *Pattern.* To find either DATE or DATA, use "DAT_" for *Pattern.*

These characters can be escaped (that is, ignored) using a \. For example, to search for a query that contains the character %, use \% in *Pattern.*

If *Pattern* is an empty string, this method retrieves all queries of the specified type (that is, specifying "" for *Pattern* is the same as specifying "%").

Parameters

Parameter	***Description***
SearchType	Specify the type of search to be used with the given pattern. You can use either a constant or a number value for this parameter. See below.
Pattern	Specify the pattern to be used when searching for records.
CaseSensitive	Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it isn't.

The values for *SearchType* can be as follows:

Numeric Value	***Constant Value***	***Description***
1	%Query_FindName	Search for records with the name matching the given pattern.
2	%Query_FindDescr	Search for records with the description matching the given pattern.
3	%Query_FindNameDescr	Search for records with either the name or the description matching the given pattern.

Returns

A reference to a QueryDBRecord collection containing zero or more records.

Example

```
Local ApiObject &MySession, &DBRecList;

&MySession = %Session;

DBRecList = &MySession.SearchQueryDBRecords(%Query_FindName, "A%", False);
```

See Also

Chapter 33, "Query Classes," FindQueryDBRecords, page 1910

SearchPrivateQueries

Syntax

SearchPrivateQueries*(QueryType,UserId,SearchType,Pattern,CaseSensitive)*

Description

The SearchPrivateQueries method returns a reference to a Query collection, filled with zero or more Private queries that match the specified *SearchType*,*UserId*,*Pattern*, and *CaseSensitive* choice

Parameters

Parameter	***Description***
QueryType	Specify the type of query to be searched for. You can specify either a constant or number value for this parameter. See below.
UserId	Specify the user Id to be used to find currently signed-on user's private queries.
SearchType	Specify the type of search to be used with the given pattern. You can use either a constant or a number value for this parameter. See below.
Pattern	Specify the pattern to be used when searching for queries.
CaseSensitive	Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it isn't.

The values for *QueryType* can be as follows:

Numeric Value	**_Constant Value_**	**_Description_**
1	%Query_Query	Find queries of the type Query.
3	%Query_DBAgent	Find queries of the type Process.
4	%Query_Role	Find queries of the type Role
10	N/A	Find queries of the type Archive.

The values for *SearchType* can be as follows:

Numeric Value	**_Constant Value_**	**_Description_**
1	%Query_FindName	Search for queries with the name matching the given pattern.
2	%Query_FindDescr	Search for queries with the description matching the given pattern.
3	%Query_FindNameDescr	Search for queries with either the name or the description matching the given pattern.

Returns

A reference to a Query collection containing zero or more queries.

Example

The following example retrieves all private queries of type Query which start with A and do a case-insensitive search, that is, get all queries starting with A or a.

```
Local ApiObject &MySession, &DBRecList;

&MySession = %Session;

DBRecList = &MySession.SearchPrivateQueries(%Query_ListQuery, %UserId, ⇒
%Query_FindName, "A%", False);
```

See Also

Chapter 33, "Query Classes," FindQueries, page 1910; Chapter 33, "Query Classes," FindQueriesDateRange, page 1911 and Chapter 33, "Query Classes," SearchPublicQueries, page 1917

SearchPublicQueries

Syntax

`SearchPublicQueries`*`(QueryType, SearchType,Pattern, CaseSensitive)`*

Description

The SearchPublicQueries method returns a reference to a Query collection, filled with zero or more Public queries that match the specified *SearchType*,*Pattern,* and *CaseSensitive* choice.

Parameters

Parameter	***Description***
QueryType	Specify the type of query to be searched for. You can specify either a constant or number value for this parameter. See below.
SearchType	Specify the type of search to be used with the given pattern. You can use either a constant or a number value for this parameter. See below.
Pattern	Specify the pattern to be used when searching for queries.
CaseSensitive	Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it isn't.

The values for *QueryType* can be as follows:

Numeric Value	***Constant Value***	***Description***
1	%Query_ListQuery	Find queries of the type Query.
3	%Query_ListDBAgent	Find queries of the type Process
4	%Query_ListRole	Find queries of the type Role
10	N/A	Find queries of type Archive

The values for *SearchType* can be as follows:

Numeric Value	***Constant Value***	***Description***
1	%Query_FindName	Search for queries with the name matching the given pattern.
2	%Query_FindDescr	Search for queries with the description matching the given pattern.

Numeric Value	*Constant Value*	*Description*
3	%Query_FindNameDescr	Search for queries with either the name or the description matching the given pattern.

Returns

A reference to a Query collection containing zero or more queries.

Example

The following example retrieves all public queries of type Query that start with A and does a case-insensitive search, that is, get all queries starting with A or a.

```
Local ApiObject &MySession, &DBRecList;

&MySession = %Session;

DBRecList = &MySession.SearchPublicQueries(%Query_ListQuery, ⇒
%Query_FindName, "A%", False);
```

See Also

Chapter 33, "Query Classes," FindQueries, page 1910; Chapter 33, "Query Classes," FindQueriesDateRange, page 1911 and Chapter 33, "Query Classes," SearchPrivateQueries, page 1915

Query Collection

A query collection is returned from the following session methods:

- AdvancedSearchQueries
- FindQueries
- FindQueriesDateRange
- SearchPrivateQueries
- SearchPublicQueries

See Also

Chapter 33, "Query Classes," AdvancedSearchQueries, page 1905

Chapter 33, "Query Classes," FindQueries, page 1910

Chapter 33, "Query Classes," FindQueriesDateRange, page 1911

Chapter 33, "Query Classes," SearchPublicQueries, page 1917

Chapter 33, "Query Classes," SearchPrivateQueries, page 1915

Query Collection Methods

In this section, we discuss each query collection method.

First

Syntax

First()

Description

The First method returns the first Query object in the Query collection.

Parameters

None.

Returns

A reference to a Query object if successful, NULL otherwise.

Example

```
&MyQuery = &MyCollection.First();
```

Item

Syntax

`Item``(`*`number`*`)`

Description

The Item method returns the Query object that exists at the *number* position in the Query collection.

Item Considerations

FindQueries and FindQueriesDateRange return both public and private queries. It depends on your database whether the private query is returned first or the public query. If you have two queries with the same name, it depends on your database whether the first use of Item returns the private or the public query.

Parameters

Parameter	***Description***
Number	Specify the position number in the collection of the Query object that you want returned.

Returns

A reference to a Query object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryColl.Count;
   &MyQuery = &QueryColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

`ItemByName``(`*`Name`*`)`

Description

The ItemByName method returns the Query object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing Query within the Query collection. If you specify an invalid name, the object is NULL. The length of this parameter is 30 characters.

Returns

A reference to a Query object if successful, NULL otherwise.

Example

```
&MyQuery = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

Description

The Next method returns the next Query object in the Query collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a Query object if successful, NULL otherwise.

Example

```
&MyQuery = &MyCollection.Next();
```

Query Collection Property

In this section, we discuss the Count query collection property.

Count

Description

This property returns the number of Query objects in the Query collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Query Class

A query object is returned from the following:

- The GetQuery session method.
- The Query collection methods First, Item, ItemByName, or Next.

See Chapter 33, "Query Classes," GetQuery, page 1912.

See Chapter 33, "Query Classes," Query Collection, page 1918.

Query Class Methods

In this section, we discuss each Query class method in alphabetical order.

AddPrompt

Syntax

AddPrompt*(PromptName)*

Description

The AddPrompt method adds a prompt with the given name to the query definition. This method returns a new QueryPrompt object that you can then use to specify details about the prompt.

Note. Prompt names are *not* checked for uniqueness. Each new prompt is just added to the list of prompts.

Parameters

Parameter	*Description*
PromptName	Specify the name of the new prompt with a string. The length of this parameter is 30 characters.

Returns

A reference to a QueryPrompt object.

See Also

Chapter 33, "Query Classes," DeletePrompt, page 1929; Chapter 33, "Query Classes," Prompts, page 1953 and Chapter 33, "Query Classes," QueryPrompt Class, page 2061

AddQuerySelect

Syntax

`AddQuerySelect``()`

Description

Use the AddQuerySelect method to add the first Query Select statement into the query definition. This method returns the instance of the newly created QuerySelect.

Parameters

None.

Returns

A reference to a QuerySelect object. If the query already contains the Main Select, it returns NULL.

See Also

Chapter 33, "Query Classes," QuerySelect Collection, page 1956

AddTrackingURL

Syntax

AddTrackingURL*(URLString)*

Description

Use this method to define the base URL used to construct a fully qualified drilling URL. A base URL consists of the following elements:

http://*server_name/servlet_name/site_name/portal_name/node_name*

For example:

http://server.mycompany.com:8080/psp/ps_2/EMPLOYEE/EMP_LOCAL

The string passed in as the base URL is not validated; therefore, it is the developer's responsibility to ensure that the value is complete and correctly formatted.

This method is used when a program does not have context to identify the site, portal, and node. For example, batch programs such as Application Engine programs do not operate in the context of a site, portal, and node. Consequently, prior to invoking an Application Engine program, the calling program needs to define and store the base URL so that the Application Engine program can use this stored value with the AddTrackingURL method.

Parameters

Parameter	*Description*
URLString	Specify the base URL as a string. The base URL is used to construct a fully qualified drilling URL.

Returns

None.

Example

The ExecQry step of the PSQUERY Application Engine program invokes the AddTrackingURL method using a value stored in the run control table prior to executing the query.

```
&aQry = %Session.GetQuery();
If &aQry.Open(PSQUERY_AET.QRYNAME, &bPublic, False) = 0 Then

   &aQry.AddTrackingURL(PSQUERY_AET.URL);
   ...
   &Result = &aQry.RunToFile(&rcdQryPrompts, &sOutFile, %OutDestFormat, 0);
End-if;
```


See Also

Chapter 33, "Query Classes," SetTrackingURL, page 1944

Chapter 33, "Query Classes," Setting a Drilling URL, page 1896

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "Defining Selection Criteria," Drilling URL in Oracle PeopleSoft Query

Close

Syntax

Close()

Description

The Close method closes the query, freeing the memory associated with that object, and discarding any changes made to the query since the last save. The Close method can be used only on an open query, not a closed query. This means you must have opened the query with the Open or Create methods before you can close it. To save any changes, you must use the Save method before using Close.

It's very important to close your query when you're finished processing. Canceling out of a page does *not* close a query. You may receive error messages every other time you run your program if you haven't closed your queries.

Parameters

None.

Returns

None.

See Also

Chapter 33, "Query Classes," Open, page 1933; Chapter 33, "Query Classes," Save, page 1944 and Chapter 33, "Query Classes," Create, page 1926

CopyPrivateQuery

Syntax

CopyPrivateQuery(*QueryName*,*QryType*,*TargetUserID*)

Description

The CopyPrivateQuery method copies the query from the current user to a user specified by the target user ID.

Parameters

Parameter	***Description***
QueryName	Name of the query to be copied, as a string.
QryType	Specify the type of query. This parameter takes either a numeric or constant value. See below.
TargetUserID	The user ID to which the query is to be copied, as a string.

The values for *QryType* can be as follows:

Numeric Value	***Constant Value***	***Description***
1	%Query_Query	Find queries of the type Query.
4	%Query_Role	Find queries of the type Role.
5	%Query_DBAgent	Find queries of the type Process.
7	%Query_Archive	Find queries of the type Archive.

Returns

0 if successful.

Create

Syntax

`Create`*`(QueryName,Public,Type, Description,LongDescription)`*

Description

The Create method creates a new query, based on the parameters passed with the method. The specified query must be a *new* query.

Warning! If you specify the name of a query that already exists, the existing query is overwritten by the new query.

The Create method can be used only with a closed query, it cannot be used on an open query.

After you create a new query, you don't have to open it with the Open method. The existing query object points to the new query.

Creating a new query doesn't create the query in the database. You must save the query (with the Save method) to commit it to the database.

Parameters

Parameter	***Description***
QueryName	Name of the query to be created. This parameter takes a string value. This parameter takes 30 characters. A query name can contain only alphabetic and numeric characters, as well as underscores.
Public	Specify if the query is public or private. This parameter takes a Boolean value: True if the query is public, False if it's a private query.
Type	Specify the type of query. This parameter takes either a numeric or constant value. See below.
Description	Specify a short description for the query. This parameter takes a string value. This parameter takes 30 characters.
LongDescription	Specify a long description for the query. This parameter takes a string value. The length of this parameter depends on your system database limit for LONG fields.

The values for *Type* can be as follows:

Numeric Value	***Constant Value***	***Description***
1	%Query_Query	Query
3	%Query_View	View
4	%Query_Role	Role
5	%Query_DBAgent	Process
7	%Query_Archive	Archive

Returns

An integer value: 0 means the query was created successfully.

Example

```
/* Use the existing session */

&MySession = %Session;

&MyQry = &MySession.GetQuery();

/* create a new query : public type-User */
&Rslt = &MyQry.Create("MYQUERY", True, %Query_Query, "My Query", ⇒
"My first Query");

If &Rslt = 0 Then
   /* Query created successfully */
Else
   /* do error processing */
End-If;
```

See Also

Chapter 33, "Query Classes," GetQuery, page 1912; Chapter 33, "Query Classes," Close, page 1925 and Chapter 33, "Query Classes," Save, page 1944

Delete

Syntax

Delete()

Description

The Delete method deletes the specified query from the database. The Delete method can be used only with an open query, it cannot be used on a closed query. Before you use the Delete method, you must explicitly open the query object to be deleted (with either the Open or Create method.)

Parameters

None.

Returns

An integer value: 0 means the query was deleted successfully.

See Also

Chapter 33, "Query Classes," Save, page 1944

DeletePrompt

Syntax

DeletePrompt*(PromptNumber)*

Description

The DeletePrompt method deletes a prompt from a query definition.

Parameters

Parameter	***Description***
PromptNumber	Specify the numeric location in the query definition of the prompt that you want to delete.

Returns

An integer value: 0 means the query was deleted successfully.

See Also

Chapter 33, "Query Classes," AddPrompt, page 1922; Chapter 33, "Query Classes," Prompts, page 1953 and Chapter 33, "Query Classes," QueryPrompt Class, page 2061

FindExpression

Syntax

FindExpression*(ExpressionNumber)*

Description

Use the FindExpression method to search the query definition for an expression with the given expression number. Although expressions are associated with the QuerySelect in which they are defined, PeopleSoft Query doesn't qualify expressions by Select number. Therefore, each expression has a unique number across all QuerySelect objects.

Parameters

Parameter	*Description*
ExpressionNumber	Specify the numeric value of the expression number to be searched for in the query definition.

Returns

A QueryExpression object if successful, NULL otherwise.

See Also

Chapter 33, "Query Classes," AddPrompt, page 1922; Chapter 33, "Query Classes," Prompts, page 1953 and Chapter 33, "Query Classes," QueryPrompt Class, page 2061

FormatBinaryResultString

Syntax

FormatBinaryResultString*(&Rowset,Output_Format, Start_Row,End_Row)*

Description

Use the FormatBinaryResultString method to generate a string containing the content of the input (the rowset) in XLS format. You can specify the range of rows to be in the rowset to be used as input. The rowset object is created using the RunToRowset method.

Parameters

Parameter	*Description*
&Rowset	Specify an already instantiated and populated rowset object containing the query result. This rowset is created using the RunToRowset method.
Output_Format	Specify the format of the output. You can specify either a numeric or constant value. See below.
Start_Row	Specify the first row in the rowset that you want to use for output. This parameter takes a numeric value.
End_Row	Specify the last row in the rowset that you want to use for output. This parameter takes a numeric value.

The values for *Output_Format* can be as follows:

Numeric Value	Constant Value	Description
8	%Query_XLS	The output for Excel in HTML format.

Returns

A binary object containing the formatted output.

Example

```
/* Use RowCount, not ActiveRowCount, to get the total number of rows. */

&End = &MyRowset.RowCount;

&FormString = &MyQuery.FormatBinaryResultString (&MyRowset, %Query_XLS, 1, &End);
```

See Also

Chapter 33, "Query Classes," RunToFile, page 1935 and Chapter 33, "Query Classes," RunToRowset, page 1938

FormatResultString

Syntax

FormatResultString*(&Rowset,Output_Format,StartRow,EndRow)*

Description

Use the FormatResultString method to generate a string containing the content of the input (the rowset) in HTML, PDF, XLS, or CSV format. You can specify the range of rows to be in the rowset to be used as input. The rowset object is created using the RunToRowset method.

Parameters

Parameter	Description
&Rowset	Specify an already instantiated and populated rowset object containing the query result. This rowset is created using the RunToRowset method.
Output_Format	Specify the format of the output. You can specify either a numeric or constant value. See below.
Start_Row	Specify the first row in the rowset that you want to use for output. This parameter takes a numeric value.

Parameter	*Description*
End_Row	Specify the last row in the rowset that you want to use for output. This parameter takes a numeric value.

The values for *Output_Format* can be as follows:

Numeric Value	*Constant Value*	*Description*
2	%Query_PDF	The output is in PDF format.
5	%Query_HTML	The output is in HTML format.
8	%Query_XLS	The output for Excel in HTML format.
14	%Query_TXT	The output for text in CSV format.
20	%Query_XML_XmlP	The output is in XMLP format.

Returns

A string containing the formatted output.

Example

```
/* Use RowCount, not ActiveRowCount, to get the total number of rows. */

&End = &MyRowset.RowCount;

&FormString = &MyQuery.FormatResultString(&MyRowset, %Query_TXT, 1, &End);
```

See Also

Chapter 33, "Query Classes," RunToFile, page 1935 and Chapter 33, "Query Classes," RunToRowset, page 1938

GetTreePromptCount

Syntax

GetTreePromptCount()

Description

Use the GetTreePromptCount method to get the number of tree prompts if available in the query.

Parameters

None.

Returns

A number that indicates the count of tree prompts used in the query. -1 in case of an error.

Example

In this example, the ExecQry step of the PSQUERY Application Engine program invokes the GetTreePromptCount() method using a value stored in the run control table prior to executing the query.

```
&aQry = %Session.GetQuery();
If &aQry.Open(PSQUERY_AET.QRYNAME, &bPublic, False) = 0 Then
   &nTreePromptCount = &aQry.GetTreePromptCount();
End-if;
```

Open

Syntax

Open*(QueryName,Public,Update*)

Description

The Open method opens the query object specified by the parameters. The Open method can be used only with a closed query, it cannot be used on an open query. You cannot read or set any properties of a query until after you open it.

Considerations for Opening Different Types of Queries

When you use the Open method, the system tries to open queries according to the type of query. The following is the order used for searching for the query to open:

1. Query (User)
2. View
3. Role
4. Process
5. Archive

All query names are unique. You can't have two queries with the same name, just of different types.

Parameters

Parameter	*Description*
QueryName	Specify the name of the query to be opened. You must specify an existing query. This parameter takes a string value.
Public	Specify if the query is public or private. This parameter takes a Boolean value: True if the query is public, False if it's a private query.
Update	This parameter is required, but is unused in this release. You must specify a either True or False.

Returns

An integer value: 0 means the query was opened successfully.

See Also

Chapter 33, "Query Classes," GetQuery, page 1912; Chapter 33, "Query Classes," Close, page 1925 and Chapter 33, "Query Classes," Save, page 1944

Rename

Syntax

`Rename``(`*`NewQueryName`*`)`

Description

The Rename method renames the existing query definition with *NewQueryName.* The Rename method can be used only with an open query, it cannot be used on a closed query. Before you use the Rename method, you must explicitly open the query object to be renamed (with either the Open or Create method.)

Note. The Rename method takes place immediately. You don't have to save the query for the rename to occur.

Parameters

Parameter	*Description*
NewQueryName	Specify the new name for the existing query. The maximum length of this parameter is 30 characters.

Returns

An integer value: 0 means the query was renamed successfully.

See Also

Chapter 33, "Query Classes," Open, page 1933; Chapter 33, "Query Classes," Close, page 1925 and Chapter 33, "Query Classes," Save, page 1944

RunToFile

Syntax

```
RunToFile(&PromptRecord,Destination,OutputFormat,MaxRows)
```

Description

Use the RunToFile method to execute the Query and return the result to the file specified with *Destination.* The query should be an existing query in the database, or it should have been saved using the Save method.

Because a Query may have runtime prompts (that is, criteria defined using Prompt), this method requires those values to be passed in when you execute this method. *PromptRecord* is a PeopleCode record object containing the prompt values as fields in the record. You can use the PromptRecord property to obtain this object.

Destination must include the absolute path name of where the file is to be created.

If the specified subdirectory does not exist, this method does *not* automatically create them for you, and you receive an error.

If you specify HTML as the output format, the PeopleSoft Query style sheet PSQUERYSTYLEDEF is used for formatting the output.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Style Sheet Definitions"

Parameters

Parameter	*Description*
&PromptRecord	Specify an instance of a PeopleCode record that contains the runtime prompts and values required for running the query.
Destination	Specify the absolute path name of where the files are to be created.

Parameter	***Description***
OutputFormat	Specify the format of the data being written to the file. You can use either a constant or a numeric value for this parameter. See below.
MaxRows	Specify the maximum number of rows to be fetched. This parameter takes a numeric value. The values are: • -1 returns all rows *regardless* of the setting on the Query security profile (MaxRowsFetched). • 0 returns the maximum number of rows allowed by the Query security profile. • >0 is the limit on the number of rows.

The values for *OutputFormat* can be as follows:

Numeric Value	***Constant Value***	***Description***
2	%Query_PDF	The output is in PDF format.
5	%Query_HTML	The output is in HTML format.
8	%Query_XLS	The output for Excel in HTML format.
14	%Query_TXT	The output for txt in CSV format.
17	%Query_XML_WebRowset	The output is in web rowset XML format.
20	%Query_XML_XmlP	The output is in XMLP format.

Returns

Returns 0 if successful.

Example

To run a query using the query API RunToFile method:

1. Open the query.

```
&aRunQry = %Session.GetQuery();
```

```
&aRunQry.Open(&sQryName, False, False);
```

2. Obtain the PromptRecord for the query.

```
&aQryPromptRec = &aRunQry.PromptRecord;
```

This instance of the PromptRecord can be passed to the PeopleCode Prompt function to prompt the user for the runtime values, as follows:

```
&nResult = Prompt(&strQryName | " Prompts", "", &aQryPromptRec);
```

3. Run the query.

Now that you have the runtime values, the query can be run, as follows:

```
&aRowSet = &aRunQry.RunToFile(&aQryPromptRec, "c:\temp\QueryOutput.html", ⇒
%Query_PDF, 0);
```

4. Access the data of the rowset.

You can now manipulate the rowset using the data buffer access methods and properties:

```
&aRowSet(&i).GetRecord(1).GetField(&j).Value
```

The following is a complete sample code example:

```
Local Rowset &aRowSet;
   Local Row &aRow;
   Local Record &aQryPromptRec;
   Local Record &aRec;
   Local ApiObject &aRunQry;
   &strHTML = "";
   &aRunQry = %Session.GetQuery();
   If (&aRunQry.Open(&sQryName, False, False) <> 0) Then
      &strHTML = "Error in opening query";
   Else
      &aQryPromptRec = &aRunQry.PromptRecord;
      &strQryName = &aRunQry.Name;
   If &aQryPromptRec <> Null Then
      &nResult = Prompt(&strQryName | " Prompts", "", &aQryPromptRec);
   End-If;
   If (&aRunQry.RunToFile(&aQryPromptRec, "c:\temp\" | ⇒
&aRunQry.Name, 3, 0) = 0) Then
      &strHTML = "Resultset saved into file successfully.";
   Else
      &strHTML = "Failed to save Resultset into file.";
   End-If;
End-If;
```

See Also

Chapter 33, "Query Classes," RunToRowset, page 1938 and Chapter 33, "Query Classes," PromptRecord, page 1953

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Prompt

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

RunToRowset

Syntax

RunToRowset*(&PromptRecord,MaxRows)*

Description

Use the RunToRowset method to execute the Query and return the result to a rowset. The query should be an existing query in the database, or it should have been saved using the Save method.

Because a Query may have runtime prompts (that is, criteria defined using Prompt), this method requires those values to be passed in when you execute this method. *PromptRecord* is a PeopleCode record object containing the prompt values as fields in the record. The PromptRecord property can be used to obtain this object.

Parameters

Parameter	***Description***
&PromptRecord	Specify an instance of a PeopleCode record that contains the runtime prompts and values required for running the query.
MaxRows	Specify the maximum number of rows to be fetched. This parameter takes a numeric value. The total number of rows fetched is the minimum value of this parameter's value, and the application server configuration parameter Max Fetch Size (which is specified in KB.) The values are: • -1 returns all rows *regardless* of the setting on the Query security profile (MaxRowsFetched). • 0 returns the maximum number of rows allowed by the Query security profile. • >0 is the limit on the number of rows.

Returns

If successful, the query result is returned as a populated PeopleCode rowset. If the query wasn't successful, the method returns NULL. If unsuccessful, check the PSMessages collection for errors.

Example

To run a query using the query API RunToRowset method:

1. Open the Query.

   ```
   &aRunQry = %Session.GetQuery();
   ```

   ```
   &aRunQry.Open(&sQryName, False, False);
   ```

2. Get the PromptRecord for the Query.

   ```
   &aQryPromptRec = &aRunQry.PromptRecord;
   ```

   This instance of the PromptRecord can be passed to the PeopleCode Prompt function to prompt the user for the runtime values, as follows:

   ```
   &nResult = Prompt(&strQryName | " Prompts", "", &aQryPromptRec);
   ```

3. Run the query.

   Now that you have the prompt values, you can run the query:

   ```
   &aRowSet = &aRunQry.RunToRowset(&aQryPromptRec, 0);
   ```

4. Access the data of the rowset.

   You can now manipulate the rowset using the data buffer access methods and properties:

   ```
   &aRowSet(&i).GetRecord(1).GetField(&j).Value
   ```

The following is a complete sample code example:

```
      Local Rowset &aRowSet;
      Local Row &aRow;
      Local Record &aQryPromptRec;
      Local Record &aRec;
      Local ApiObject &aRunQry;
      &strHTML = "";
      &aRunQry = %Session.GetQuery();
      If (&aRunQry.Open(&sQryName, False, False) <> 0) Then
         &strHTML = "Error in opening query";
      Else
         &aQryPromptRec = &aRunQry.PromptRecord;
         &strQryName = &aRunQry.Name;
      If &aQryPromptRec <> Null Then
         &nResult = Prompt(&strQryName | " Prompts", "", &aQryPromptRec);
         &aRowSet = &aRunQry.RunToRowset(&aQryPromptRec, 0);
            If &aRowSet <> Null Then
               &strHTML = &strHTML | "<table cellpadding='2' ⇒
   cellspacing='0' width='90%'>";
               &aRec = &aRowSet(1).GetRecord(1);
               &QrySelOutputFldCol = &aRunQry.QuerySelect.QueryOutputFields;
               If &QrySelOutputFldCol <> Null Then
                  &strHTML = &strHTML | "<TR><TD>";
                  &strHTML = &strHTML | "<table border='1' cellpadding='2'⇒
    cellspacing='0' width='100%'>";
                  &strHTML = &strHTML | "<TR><TH> </TH>";
                  For &j = 1 To &QrySelOutputFldCol.count
                     &strHTML = &strHTML | "<TH><B>" | &QrySelOutputFldCol.Item⇒
   (&j).LongName | "</B></TH>";
                  End-For;
                  &strHTML = &strHTML | "</TR>";
                  &strHTML = &strHTML | "</TD></TR>";
               End-If;
               &strHTML = &strHTML | "<TR><TD>";
               If &aRowSet.RowCount > 0 Then
                  For &i = 1 To &aRowSet.RowCount
                     &aRow = &aRowSet(&i);
                     &strHTML = &strHTML | "<TR>" | "<TD>" | &i | "</TD>";
                     For &j = 1 To &aRow.GetRecord(1).FieldCount
                        &strHTML = &strHTML | "<TD>" | &aRow.GetRecord(1).GetField⇒
   (&j).Value | "</TD>";
                     End-For;
                     &strHTML = &strHTML | "</TR>";
                  End-For;
               Else
                  &strHTML = &strHTML | "No records retrieved.";
               End-If;
               &strHTML = &strHTML | "</TD></TR>";
               &strHTML = &strHTML | "</TABLE>";
               &strHTML = &strHTML | "</TABLE>";
            Else
               &strHTML = "Failed to retrieve result set.";
            End-If;
         End-If;
      End-If;
```


See Also

Chapter 33, "Query Classes," RunToFile, page 1935 and Chapter 33, "Query Classes," PromptRecord, page 1953

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Prompt

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

RunToString

Syntax

`RunToString``(&PromptRecord,ChunkSize,OutputFormat,MaxRows)`

Description

Use the RunToString method to execute the query and return the result as a formatted string.

When "chunking" is active, RunToString is intended to be called in a recursive fashion until the result set has been completely traversed. A Boolean property, MoreRowsAvailable, is included in the Query class to control recursive execution of this method.

See Chapter 33, "Query Classes," MoreRowsAvailable, page 1951.

Parameters

Parameter	***Description***
&PromptRecord	Specify an instance of a PeopleCode record that contains the runtime prompts and values required for running the query.
ChunkSize	Specify the desired chunking size in characters as a number. 0 means no chunking.
OutputFormat	Specify the format of the data being written to the file. You can use either a constant or a numeric value for this parameter. See below.
MaxRows	Specify the maximum number of rows to be fetched. This parameter takes a numeric value. The values are: • -1 returns all rows regardless of the setting on the query security profile (MaxRowsFetched). • 0 returns the maximum number of rows allowed by the query security profile. • >0 is the absolute limit on the number of rows.

The values for *Output_Format* can be as follows:

Numeric Value	*Constant Value*	*Description*
2	%Query_PDF	The output is in PDF format.
5	%Query_HTML	The output is in HTML format.
8	%Query_XLS	The output for Excel in HTML format.
14	%Query_TXT	The output for text in CSV format.
17	%Query_XML_WebRowset	The output is in web rowset XML format.
20	%Query_XML_XmlP	The output is in XMLP format.

Returns

A string containing the formatted output. If an error occurs, an empty string will be returned.

If there are no errors, but no data is in the query result set, the string will have system-defined header and footer information, but no rows will be present.

Example

The following example runs a query without chunking.

```
&queryname = "XRFWIN";

rem &querytype = %Query_XLS;
rem &querytype = %Query_PDF;
rem &querytype = %Query_HTML;
rem &querytype = %Query_TXT;
&querytype = %Query_XML_XmlP;
rem &querytype = %Query_XML_WebRowset;


&aRunQry = %Session.GetQuery();
If (&aRunQry.Open(&queryname, False, False) <> 0) Then
   MessageBox(0, "", 0, 0, "Error opening query");
Else
   &aQryPromptRec = &aRunQry.PromptRecord;

   &chunksize = 0;

   &filename = "C:\QueryWork850\output\" | &querytype | "_runtostring.xml";

   &resultString = &aRunQry.RunToString(&aQryPromptRec, &chunksize, &querytype, 0);

   &myfile = GetFile(&filename, "w", %FilePath_Absolute);
   &myfile.writestring(&resultString);
   &myfile.close();

   MessageBox(0, "", 0, 0, "Success!");

End-If;
```


The following example runs a query with chunking.

```
&queryname = "XRFWIN";
rem &querytype = %Query_XLS;
rem &querytype = %Query_PDF;
rem &querytype = %Query_HTML;
rem &querytype = %Query_TXT;
&querytype = %Query_XML_XmlP;
rem &querytype = %Query_XML_WebRowset;


&aRunQry = %Session.GetQuery();
If (&aRunQry.Open(&queryname, False, False) <> 0) Then
   MessageBox(0, "", 0, 0, "Error opening query");
Else
   &aQryPromptRec = &aRunQry.PromptRecord;
   &counter = 0;

   &chunksize = 1000;

   Repeat
      &counter = &counter + 1;
      &filename = "C:\QueryWork850\output\" | &querytype | &counter | ⇒
"_runtostring.xml";

      &resultString = &aRunQry.RunToString(&aQryPromptRec, &chunksize, ⇒
&querytype, 0);

      If (Len(&resultString) > 0) Then

         &myfile = GetFile(&filename, "w", %FilePath_Absolute);
         &myfile.writestring(&resultString);
         &myfile.close();

      Else

         /* edge condition; if there are no more rows AND the */
         /* returned string is empty this is not an error.    */

         If (&aRunQry.MoreRowsAvailable) Then
            /* we have an error... Yikes! */
            MessageBox(0, "", 0, 0, "Something bad has happened!");
         Else
            /* no worries, just ignore it */
         End-If;

      End-If
   Until (Not &aRunQry.MoreRowsAvailable);

   MessageBox(0, "", 0, 0, "Success!");

End-If;
```

See Also

Chapter 33, "Query Classes," MoreRowsAvailable, page 1951

Save

Syntax

`Save``()`

Description

The Save method writes any changes to the existing query to the database.

The Save method can be used only on an open query, not on a closed query. This means you must have opened the query with the Open method before you can save it.

The query object remains open after executing Save. You must execute the Close method on the object before it is closed and the memory freed.

Note. If you're calling the query API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

Parameters

None.

Returns

An integer value: 0 means the query was saved successfully.

See Also

Chapter 33, "Query Classes," Open, page 1933 and Chapter 33, "Query Classes," Close, page 1925

SetTrackingURL

Syntax

`SetTrackingURL``(`*`ExpressionText,ExpressionNumber`*`)`

Description

Use the SetTrackingURL method to re-establish the drilling URL if the current execution context is different from the context in which the drilling URL was initially set.

For example, a drilling URL is set as a query expression by the program that executes the query. After query execution, a different program—an iScript program—allows the user to download the query results to an Excel spreadsheet. This iScript program needs to include the drilling URL in the spreadsheet data. In order for the iScript program to have access to the drilling URL query expression values, these values must be defined as global objects by the program that executes the query. Then, the iScript program can re-establish the drilling URL by calling the SetTrackingURL method.

Parameters

Parameter	***Description***
ExpressionText	Specify the drilling URL as a string by reference to an established query expression.
ExpressionNumber	Specify the drilling URL as a numeric value by reference to an established query expression.

Returns

None.

Example

```
If &rsURLList <> Null And
      &rsURLList.ActiveRowCount >= 1 And
      All(&rsURLList(1).QRY_URL_WRK.QRYCRIT1EXPRTEXT.Value) Then
   For &nCount = 1 To &rsURLList.ActiveRowCount;
      &rRecordExpr = &rsURLList(&nCount).QRY_URL_WRK;
      &QryObj.SetTrackingURL(&rRecordExpr.QRYCRIT1EXPRTEXT.Value, ⇒
&rRecordExpr.QRYCRIT1EXPRNUM.Value);
   End-For;
End-If;
```

See Also

Chapter 33, "Query Classes," AddTrackingURL, page 1924

Chapter 33, "Query Classes," Setting a Drilling URL, page 1896

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "Defining Selection Criteria," Drilling URL in Oracle PeopleSoft Query

Query Class Properties

In the following section, we discuss each Query class property.

Approved

Description

This property returns a string indicating whether a query is approved, unapproved, or modified. This is useful for query administration. The values are:

Value	***Description***
U	Query is unapproved. This is the default value. The query administrator can prevent execution of unapproved queries.
A	Query has been approved by the query administrator.
M	Query has been modified.

This property is read-write.

ApproveOprId

Description

Note. This property has been deprecated, and remains for backward compatibility only. Use the ApproveUserId property instead.

This property returns a string containing the user ID of the user who approved the query. This can be useful for query administration.

This property is read-write.

See Also

Chapter 33, "Query Classes," ApproveUserId, page 1946

ApproveUserId

Description

This property returns a string containing the user ID of the user who approved the query. This can be useful for query administration.

This property is read-write.

ApproveDtTm

Description

This property returns the date-time stamp when the query was most recently approved. This can be useful for query administration.

This property is read-only.

CreateOprId

Description

Note. This property has been deprecated, and remains for backward compatibility only. Use the CreateUserId property instead.

This property returns a string containing the User Id of the user who created the query. This can be useful for query administration.

This property is read-only.

See Also

Chapter 33, "Query Classes," CreateUserId, page 1947

CreateDtTm

Description

This property returns a string containing the date-time stamp indicating when the query was created. This property is set by the query runtime when a new query is saved. This can be useful for query administration.

This property is read-only.

CreateUserId

Description

This property returns a string containing the User Id of the user who created the query. This can be useful for query administration.

This property is read-only.

Description

Description

This property returns or sets the short description for the query.

The length of this property is 30 characters.

This property is read-write.

Disabled

Description

This property indicates if the query is active or not. This property returns a string value:

Value	***Description***
Y	This query is not active
Blank or any other value	This query is active

This property is read-write.

ExecAppName

Description

This property specifies the name of the application executing the query. This property isn't required. This means this property is used only if the query will be executed. This name is stored in the query execution log. This property returns a blank value when the query is opened.

When executing a query using the query API, this property should be set to the Application name that's executing it, so that the Query Monitor can track query execution. Doing this makes this property useful for query administration. If you try to read this property before setting it, you receive a NULL string.

This property should be set before using RunToRowset or RunToFile.

This property is read-write. However, this property is generally used only to set the name, rather than to read it.

ExecLogging

Description

This property indicates whether an execution log should be created when the query is executed. This can be useful for query administration. This property takes a Boolean value: True, create a log, False, don't create one.

The execution log, which is created by setting this property, can be viewed from the Query Monitor. The logging is done in a PeopleSoft Table. It stores, along with other relevant information, the Execution DateTime, Total Execution Time for the query, and Total Fetch Time for the query.

If you try to read this property before setting it, you receive a NULL string.

This property is read-write. However, this property is generally used only to set logging, rather than read.

Folder

Description

This property is used to group queries together. This parameter takes a string value, up to 18 characters.

This property is read-write.

LastSQLErrorCode

Description

This property returns the last SQL error code returned by the database as a number. The session object contains the error text and any other errors encountered while executing the query.

PeopleSoft recommends checking this value after using RunToRowset or RunToFile.

This property is read-only.

LastUpdDttm

Description

This property returns the most recent date-time when the query was updated as a string.

This property is read-only.

LastUpdOprId

Description

This property returns the User Id of the user who updated the query most recently as a string.

This property is read-only.

LongDescription

Description

This property returns or sets the long description for the query.

The length of this property is 256 characters.

This property is read-write.

Metadata

Description

This property returns a metadata collection.

This property is read-only.

Example

```
&MetadataList = &MyQuery.Metadata;
```

See Also

Chapter 33, "Query Classes," Query Metadata Collection, page 2031

MetaSQL

Description

This property returns the SQL that represents the query, unresolved for any platform.

For example, the MetaSQL property returns the following:


```
SELECT %DATEOUT(A.ASOF_DT)
 FROM PS_AEREQUESTTBL A
 WHERE A.ASOF_DT = %DATEIN('1900-01-01')
```

This property is read-only.

MoreRowsAvailable

Description

This property returns a Boolean value indicating whether "chunking" is turned on for the query.

This property is read-only.

Chunking Considerations

Every time a row is retrieved from the query result set, it is added to the internal output string managed by RunToString. At this point, the method can check to see if chunking is active. If so, and if the current size of the output string is larger than the ChunkSize parameter, then the query context is stored and the output string is returned. If not, then the next row is retrieved from the result set and the process continues.

When chunking is active, RunToString is intended to be called in a recursive fashion until the result set has been completely traversed. A Boolean property, MoreRowsAvailable, is included in the Query class to control recursive execution of these methods.

See Also

Chapter 33, "Query Classes," RunToString, page 1941

Name

Description

This property returns the name of the query as a string.

The length of this property is 30 characters.

This property is read-only.

OutputUnicode

Description

Use this property to set or return a Boolean value indicating whether the RunToFile method will create a Unicode-encoded CSV file as output. This property is applicable only if the output format is set to %Query_TXT. Because the default value is false, OutputUnicode must be set before calling RunToFile.

Note. Because Microsoft Excel does not support UTF-8 encoding, the CSV file is written in binary mode with UCS-2 encoded data. Moreover, Excel does not automatically recognize Unicode-encoded, comma-delimited files even if they have a .csv extension. Therefore, the user receiving the file will not be able to open it by double-clicking. Instead, he or she must open it with Excel's File, Open menu and choose the comma delimiter.

This property is read-write.

Example

```
&aQry.OutputUnicode = True;
&Result = &aQry.RunToFile(&rcdQryPrompts, &sOutFile, %Query_TXT, 0);
```

See Also

Chapter 33, "Query Classes," RunToFile, page 1935

PDFFont

Description

This property is used to set the PDF Font number required for generating PDF using RunToFile. This property takes either a numeric or constant value. The values are:

Numeric Value	*Constant Value*	*Description*
0	%PDFFont_None	Default value
2	%PDFFont_TraditionalChinese	The output must be written using the Traditional Chinese Font.
3	%PDFFont_SimplifiedChinese	The output must be written using the Simplified Chinese Font.
1	%PDFFont_Japanese	The output must be written using the Japanese Font
4	%PDFFont_Korean	The output must be written using the Korean Font

If you try to read this property before setting it, you receive a NULL string.

This property is read-write.

See Also

Chapter 33, "Query Classes," RunToFile, page 1935 and Chapter 33, "Query Classes," RunToRowset, page 1938

Prompts

Description

This property returns all the prompts defined for the Query in a QueryPrompt Collection. If you just want the prompts used in the criteria, use the RunTimePrompts property.

This property is read-only.

See Also

Chapter 33, "Query Classes," RunTimePrompts, page 1954 and Chapter 33, "Query Classes," QueryPrompt Collection, page 2058

PromptRecord

Description

This property returns the runtime prompts of a query as an instance of a PeopleCode record object. This can be used with the Prompt built-in function to prompt for bind values. This record instance can also be used as the first input parameter for the RunToRowset and RunToFile methods.

This property is read-only.

See Also

Chapter 33, "Query Classes," RunToFile, page 1935 and Chapter 33, "Query Classes," RunToRowset, page 1938

PublicPrivate

Description

This property specifies whether the query is public or private.

You can use either a constant or numeric value for this property. The values are:

Numeric Value	*Constant Value*	*Description*
0	%Query_Private	The query is a private query.
1	%Query_Public	The query is a public query.

This property is read-write.

Example

```
If &QryObj.PublicPrivate = %Query_Public Then
   /* code when working with Public Query */
Else
   /* code when working with Private Query */
End-if;
```

QuerySelect

Description

This property returns the Main Select (that is, the first select) of the query definition as a QuerySelect object. For a new query which does not have any selects, it returns NULL.

The Query Records, Query Criteria, QueryOutput fields, and QuerySelected Fields can be obtained using the QuerySelect object only. Hence, after opening a query, this property serves as the starting point for getting the different components of the Select statement.

This property is read-only.

See Also

Chapter 33, "Query Classes," QuerySelect Class, page 1960

QueryStatistics

Description

This property returns statistical information pertaining to the query's execution as a QueryStatistics object.

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryStatistics Class, page 2036

RunTimePrompts

Description

This property returns the prompts used in the Query Definition's criteria in a QueryPrompt Collection. To return all the prompts defined for the query, use the Prompts property.

This is property is read-only.

See Also

Chapter 33, "Query Classes," Prompts, page 1953 and Chapter 33, "Query Classes," QueryPrompt Collection, page 2058

SQL

Description

This property returns the SQL statement as generated from the query definition as a character string.

Note. The value of the SQL property is never stored as part of the query definition. Instead, it's generated by the system whenever it's needed for one of the RunTo methods or for the SQL property.

In the PeopleSoft Query, this is located under the SQL tab.

This property is read-only.

Considerations Using the SQL Property

This property returns a basic SQL statement. This statement does not include logic for doing related language record transactions or translate descriptions. PeopleSoft does not recommend using this SQL for selecting data. Instead, PeopleSoft recommends using the RunToRowset method for selecting data.

Type

Description

This property specifies the type of query. This parameter takes either a constant or number value. Values are:

Numeric Value	***Constant Value***	***Description***
1	%Query_Query	Query of type Query
3	%Query_View	Query of type View
4	%Query_Role	Role
5	%Query_DBAgent	Process
7	%Query_Archive	Archive

This property is read-write.

QuerySelect Collection

A QuerySelect collection is returned from the QuerySelects property of each QuerySelect instance. It contains the unions and subqueries, which are children of the current select statement. PeopleSoft Query allows unions only for the main select. The QuerySelect instance for any subqueries cannot have any unions.

See Chapter 33, "Query Classes," QuerySelects, page 1972.

QuerySelect Collection Methods

In this section, we discuss the QuerySelect collection methods. The methods are discussed in alphabetical order.

AddUnion

Syntax

AddUnion()

Description

The AddUnion method adds a new QuerySelect object of type Union in the current QuerySelect collection. You can use AddUnion only with the first QuerySelect Collection because PeopleSoft Query doesn't' support Unions for subqueries. All unions are considered children of the Main Select.

Parameters

None.

Returns

A reference to a QuerySelect object if successful. If the current QuerySelect Collection does not belong to the first Select statement, it returns NULL. If it fails, it returns NULL.

Example

```
&QryMainSel = &Query.QuerySelect;
&QryMainSelCol = &QryMainSel.QuerySelects;
&QryUnion = &QryMainSelCol.AddUnion();
```

DeleteQuerySelect

Syntax

DeleteQuerySelect*(SelNumber)*

Description

The DeleteQuerySelect method deletes the QuerySelect instance represented *SelNumber*.

Parameters

Parameter	*Description*
SelNumber	Specify the numeric value containing the Select Number of the QuerySelect that you want to delete.

Returns

Returns 0 if successful.

Example

```
&QryMainSel = &Query.QuerySelect;
&QryMainSelCol = &QryMainSel.QuerySelects;
&QryMainSelCol.DeleteQuerySelect(3);
```

First

Syntax

First()

Description

The First method returns the first child QuerySelect object in the current QuerySelect's collection.

Parameters

None.

Returns

A reference to a QuerySelect object if successful, NULL otherwise.

Example

```
&MyChildQrySel = &MySelCollection.First();
```

Item

Syntax

Item*(number)*

Description

The Item method returns the QuerySelect object that exists at the *number* position in the current QuerySelect collection.

Parameters

Parameter	***Description***
Number	Specify the position number in the collection of the QuerySelect object that you want returned.

Returns

A reference to a QuerySelect object if successful, NULL otherwise.

Example

```
For &I = 1 to &QrySelCol.Count;
   &MyChildQrySel = &QrySelCol.Item(&I);
   /* do processing */
End-For;
```

ItemBySelNum

Syntax

ItemBySelNum*(SelNumber)*

Description

The ItemBySelNum method returns the QuerySelect object specified by *the Select Number.* Each Select Statement in a Query Definition is identified by a unique select number.

Parameters

Parameter	*Description*
SelNumber	Specify the numeric value of the Select Number.

Returns

A reference to a QuerySelect object if successful, NULL otherwise.

Example

```
&MyChildQuerySelect = &MySelCol.ItemBySelNum(3);
```

Next

Syntax

Next()

Description

The Next method returns the next QuerySelect object in the current QuerySelect collection. You can use this method only after you have used the First method: otherwise the system returns a NULL.

Parameters

None.

Returns

A reference to a QuerySelect object if successful, NULL otherwise.

Example

```
&MyChildQuerySelect = &MySelCol.Next();
```


QuerySelect Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the total number of QuerySelect objects in the current QuerySelect Collection, as a number.

This property is read-only.

Example

```
&COUNTCHILDSELS = &MYSELCOL.Count;
```

QuerySelect Class

The QuerySelect class represents the select statement of the SQL used in the query definition. This can be any of the following:

- The main select statement.
- The select statement in each union.
- The select statement in each subquery.

Consider the following SQL statement:

```
select ACCOUNT_NUM from GL_ACCOUNT_TBL_00

where PRODUCT_ID in (select DISTINCT PRODUCT_ID FROM ORDER_TBL)

union

select ACCOUNT_NUM from GL_ACCOUNT_TBL_01
```

In this example, there are three select statements. The first select statement is the main statement. The second select statement is a subquery, which is then used in the criterion for the first select statement. The union contains the third select statement. There are records and fields associated with each select statement. In the query API, the first select is accessible using Query.QuerySelect, which can be obtained as shown:

```
&MainSelect = &Qry.QuerySelect;
```

The union can be obtained from the main select, as shown:


```
/* There can be more than one union, hence there is collection of selects */
/* in the main select */

&Union1 = &MainSelect.QuerySelects.Item(1);
```

The subquery is obtained from the main select as shown:

```
/* belongs to first criteria of the select */

&SubQry1 = &MainSelect.Criteria.Item(1).Expr2SubQuery;
```

Therefore, there is one QuerySelect instance for the main select of the query, one instance per union defined in the query, and one instance per subquery.

In PeopleSoft Query, the first select (that is, the main select) is the parent of all the unions and subqueries. The main select is obtained with the QuerySelect property. The unions and subqueries are obtained from the QuerySelects property of each QuerySelect instance.

A QuerySelect object is returned from the following:

- The QuerySelect collection methods AddUnion, First, Item, ItemBySelNum, or Next
- The QuerySelect Query property

See Chapter 33, "Query Classes," QuerySelect Collection, page 1956 and Chapter 33, "Query Classes," QuerySelects, page 1972.

QuerySelect Methods

In this section, we discuss each QuerySelect method. The methods are arranged in alphabetical order.

AddAllFields

Syntax

AddAllFields*(QueryRecord)*

Description

The AddAllFields method adds all the fields in the specified query record to the query definition, that is, makes them all QueryOutputFields. The query record must already be a part of the query definition.

Parameters

Parameter	***Description***
QueryRecord	Specify the instance of an existing record in the query from which you want to add all the fields from to the query definition. This instance can be obtained from the QueryRecords collection of the QuerySelect.

Returns

An integer: 0 if successfully added.

See Also

Chapter 33, "Query Classes," AddQueryOutputField, page 1964; Chapter 33, "Query Classes," AddQueryRecord, page 1965 and Chapter 33, "Query Classes," AddQuerySelectedField, page 1965

AddCriteria

Syntax

`AddCriteria`*`(Name)`*

Description

The AddCriteria method adds new criterion to the query definition. This method returns a reference to a new QueryCriteria object that you can then use to specify details about the criteria.

Note. If you do not specify a unique name for *Name* when adding a criteria, a criteria object referencing the new criteria is returned, not the existing criteria object.

Parameters

Parameter	***Description***
Name	Specify a string containing the name of the criteria that you want to add. The maximum length of this parameter is 30 characters.

Returns

A reference to a QueryCriteria object.

See Also

Chapter 33, "Query Classes," DeleteCriteria, page 1966; Chapter 33, "Query Classes," Criteria, page 1970 and Chapter 33, "Query Classes," QueryCriteria Class, page 1998

AddExpression

Syntax

`AddExpression`*`(Name)`*

Description

The AddExpression method adds an expression to the query definition. It returns a reference to a new QueryExpression object you can use to specify details about the expression.

Parameters

Parameter	***Description***
Name	Specify a string containing the name of the expression that you want to add. This maximum length of this parameter is 30 characters.

Returns

A reference to a QueryExpression object.

See Also

Chapter 33, "Query Classes," FindExpression, page 1929; Chapter 33, "Query Classes," DeleteExpression, page 1967; Chapter 33, "Query Classes," Expressions, page 1970 and Chapter 33, "Query Classes," QueryExpression Class, page 2018

AddHavingCriteria

Syntax

`AddHavingCriteria`*`(Name)`*

Description

The AddHavingCriteria method adds new criterion to the query definition, which is intended for use in the HAVING clause of the SELECT statement. This method returns a reference to a new QueryCriteria object that you can then use to specify details about the having criteria.

Note. If you do not specify a unique name for *Name* when adding a criterion, a criteria object referencing the new criteria is returned, not the existing criteria object.

Parameters

Parameter	*Description*
Name	Specify a string containing the name of the criteria that you want to add. The maximum length of this parameter is 30 characters.

Returns

A reference to a QueryCriteria object.

See Also

Chapter 33, "Query Classes," DeleteHavingCriteria, page 1968; Chapter 33, "Query Classes," HavingCriteria, page 1971 and Chapter 33, "Query Classes," QueryCriteria Class, page 1998

AddQueryOutputField

Syntax

`AddQueryOutputField(QueryRecord,index)`

Description

The AddQueryOutputField method adds a query output field to the query definition. This method returns a reference to a new QueryOutputField object that you can then use to specify attributes for the query definition.

Parameters

Parameter	*Description*
QueryRecord	Specify the QueryRecord instance that contains the field that you want to include in the query definition. This instance can be obtained from the QueryRecords collection of the QuerySelect.
Index	Specify the numeric position in the QueryRecord of the field that you want to add.

Returns

A reference to a QueryOutputField object.

See Also

Chapter 33, "Query Classes," AddAllFields, page 1961 and Chapter 33, "Query Classes," QueryField Class, page 1984

AddQueryRecord

Syntax

`AddQueryRecord``(`*`QueryRecordName`*`)`

Description

The AddQueryRecord method adds a query record to the query definition. You must specify a valid record name in the string parameter *QueryRecordName*. This method returns a reference to the record as a QueryRecord.

Parameters

Parameter	***Description***
QueryRecordName	Specify a string containing the name of a record to be added to the query definition. You must specify a valid record name. The maximum allowed length for this parameter is 15 characters.

Returns

A reference to the record as a QueryRecord object.

See Also

Chapter 33, "Query Classes," QueryRecord Class, page 1976

AddQuerySelectedField

Syntax

`AddQuerySelectedField``(`*`QueryRecordName,RecordAlias, FieldName, Heading`*`)`

Description

The AddQuerySelectedField method adds a query field to the query definition, as a QuerySelectedField. This method returns a reference to a new QuerySelectedField object that you can then use to specify attributes for the query definition. At the time it's added, it isn't an output field. This can be changed by setting the column number of the field to a value greater than zero.

Parameters

Parameter	***Description***
QueryRecordName	Specify the name of the query record to which the field that you want to add belongs.
RecordAlias	Specify the alias of the record (such as A, B, C, and so on.) The length of this parameter is 1.
FieldName	Specify the name of the field that's being added. The maximum allowed length for a field name is 18 characters.
Heading	Specify the heading of the field that's being added. The maximum allowed length for a heading is 30 characters.

Returns

A reference to a QuerySelectedField object.

See Also

Chapter 33, "Query Classes," AddAllFields, page 1961; Chapter 33, "Query Classes," DeleteField, page 1968 and Chapter 33, "Query Classes," QueryField Class, page 1984

DeleteCriteria

Syntax

`DeleteCriteria`*`(index)`*

Description

The DeleteCriteria method deletes the criteria specified by *index* from the query definition.

Parameters

Parameter	*Description*
Index	Specify the numeric position in the query definition of the criteria that you want to delete.

Returns

An integer: 0 if successfully deleted.

See Also

Chapter 33, "Query Classes," AddCriteria, page 1962 and Chapter 33, "Query Classes," Criteria, page 1970

DeleteExpression

Syntax

`DeleteExpression`*`(index)`*

Description

The DeleteExpression method deletes the specified expression from the query definition.

Parameters

Parameter	*Description*
Index	Specify the numeric position of the expression in the query definition that you want to delete.

Returns

An integer: 0 if successfully deleted.

See Also

Chapter 33, "Query Classes," AddExpression, page 1963; Chapter 33, "Query Classes," Expressions, page 1970 and Chapter 33, "Query Classes," QueryExpression Class, page 2018

DeleteField

Syntax

`DeleteField`*`(index)`*

Description

The DeleteField method deletes the selected field from the query definition. This does *not* delete the field from the QueryRecord, just from the query definition, so it's no longer selected.

Parameters

Parameter	***Description***
Index	Specify the position number of the field that you want deleted from the query definition.

Returns

An integer: 0 if successfully deleted.

See Also

Chapter 33, "Query Classes," AddAllFields, page 1961 and Chapter 33, "Query Classes," QueryField Class, page 1984

DeleteHavingCriteria

Syntax

`DeleteHavingCriteria`*`(index)`*

Description

The DeleteHavingCriteria method deletes the having criteria specified by *index* from the query definition.

Parameters

Parameter	*Description*
Index	Specify the numeric position in the query definition of the having criteria that you want to delete.

Returns

An integer value: 0 means the having criteria was deleted successfully.

See Also

Chapter 33, "Query Classes," AddHavingCriteria, page 1963 and Chapter 33, "Query Classes," HavingCriteria, page 1971

DeleteRecord

Syntax

`DeleteRecord(index)`

Description

The DeleteRecord method deletes the selected record from the query definition. This does *not* delete the record from the database or the QueryDBRecord collection, just from the query definition, so it's no longer selected.

Parameters

Parameter	*Description*
Index	Specify the position number of the record you want deleted from the query definition.

Returns

An integer: 0 if successfully deleted.

See Also

Chapter 33, "Query Classes," AddQueryRecord, page 1965 and Chapter 33, "Query Classes," QueryField Class, page 1984

QuerySelect Properties

In this section, we discuss the QuerySelect properties. The properties are discussed in alphabetical order.

Criteria

Description

This property returns a reference to a QueryCriteria Collection for the simple criteria of the SELECT (that is, the criteria that are used in the where clause of the select statement.)

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryCriteria Collection, page 1994

Distinct

Description

This property specifies if the Select is distinct or not.

This property takes a Boolean value: True if the query is distinct, False if the query isn't distinct.

This property is read-write.

Expressions

Description

This property returns a reference to a QueryExpression Collection.

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryExpression Collection, page 2015

HavingCriteria

Description

This property returns a reference to a QueryCriteria collection populated with Having criteria.

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryCriteria Collection, page 1994

ParentSelectNum

Description

This property returns the select number of the parent select of the current select object. Every QuerySelect is assigned a unique number indicating its place in the hierarchy of select statements. For the Main Select, this number is zero. For unions, this number is 1 (unions aren't allowed in subqueries.)

This property is read-only.

QueryOutputFields

Description

This property returns a reference to a QueryField Collection made up of QueryOutputFields.

The collection of QueryOutputFields does *not* necessarily include all columns returned in query resultset. This collection only includes columns that have been added to query output collection using the query classes or by an end-user designing a query.

PeopleSoft Query can also add additional columns for related language processing and also for translate labels on fields, and so you cannot use the Query Output Collections as a way to discover all of the columns that are returned in the resultset or by executing the SQL generated from a query.

This is property is read-only.

See Also

Chapter 33, "Query Classes," QueryField Collection, page 1980

QueryRecords

Description

This property returns a reference to a QueryRecord Collection for the query.

This is property is read-only.

See Also

Chapter 33, "Query Classes," QueryRecord Collection, page 1973

QuerySelectedFields

Description

This property returns a reference to a QueryField Collection made up of QuerySelectedFields.

This is property is read-only.

See Also

Chapter 33, "Query Classes," QueryField Collection, page 1980

QuerySelects

Description

This property returns all the QuerySelect objects that are children of the current select statement as a QuerySelect collection. If the query contains unions, this property contains unions and subqueries (if any criteria contain subqueries). For all other selects, this property contains only the subqueries (because PeopleSoft Query doesn't allow unions for sub-queries).

This is property is read-only.

See Also

Chapter 33, "Query Classes," QuerySelect Collection, page 1956

SelectNum

Description

This property returns the select number of the current select. Every QuerySelect is assigned a unique number indicating its place in the hierarchy of select statements. For the Main Select, this number would be 1.

This property is read-only.

SelectType

Description

This property specifies what type of select statement. You can check for either a numeric or a constant value. The values are:

Numeric Value	***Constant Value***	***Description***
1	%Query_SelectMain	This is the Main Select of the query definition.
2	%Query_SubQuery	This is a subquery in the query definition.
3	%Query_Union	This is a Union in the query definition.

This property is read-only.

QueryRecord Collection

A QueryRecord collection is returned from the QueryRecords QuerySelect class method.

See Chapter 33, "Query Classes," QueryRecords, page 1972.

QueryRecord Collection Methods

In this section, we discuss each QueryRecord collection method. The methods are discussed in alphabetical order.

First

Syntax

`First``()`

Description

The First method returns the first QueryRecord object in the QueryRecord collection.

Parameters

None.

Returns

A reference to a QueryRecord object if successful, NULL otherwise.

Example

```
&MyQueryRecord = &MyCollection.First();
```

Item

Syntax

`Item``(`*`number`*`)`

Description

The Item method returns the QueryRecord object that exists at the *number* position in the QueryRecord collection.

Parameters

Parameter	***Description***
Number	Specify the position number in the collection of the QueryRecord object that you want returned.

Returns

A reference to a QueryRecord object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryRecordColl.Count;
   &MyQueryRecord = &QueryRecordColl.Item(&I);
   /* do processing */
End-For;
```

ItemByAlias

Syntax

`ItemByAlias`*`(Alias)`*

Description

The ItemByAlias method returns the QueryRecord object specified by *Alias.*

Parameters

Parameter	***Description***
Alias	Specify the record alias (for example, A, B, C, and so on), used in the SQL statement as a TableName alias.

Returns

A reference to a QueryRecord object if successful, NULL otherwise.

Example

```
&MyQueryRecord = &MyCollection.ItemByAlias("A");
```

Next

Syntax

`Next``()`

Description

The Next method returns the next QueryRecord object in the QueryRecord collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryRecord object if successful, NULL otherwise.

Example

```
&MyQueryRecord = &MyCollection.Next();
```

QueryRecord Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryRecord objects in the QueryRecord Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryRecord Class

A QueryRecord object is returned from the following:

- The QueryRecord Collection methods First, Item, ItemByName, or Next.
- The AddQueryRecord QuerySelect class method.
- The QueryRecord QueryField class method.

See Chapter 33, "Query Classes," QueryRecord Collection, page 1973; Chapter 33, "Query Classes," AddQueryRecord, page 1965 and Chapter 33, "Query Classes," QueryRecord, page 1991.

QueryRecord Class Method

In this section, we discuss the GetField method.

GetField

Syntax

`GetField`*`(Index)`*

Description

The GetField method returns the QueryField object specified by *index* from the QueryRecord collection.

Parameters

Parameter	***Description***
Index	Specify the number of the field to be accessed.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&QryFieldColl = &QryRec.QueryFields;

For &I = 1 to &QryFieldColl.Count
   &QryField = &QryRec.GetField(&I);
   /* do processing */
End-For;
```

QueryRecord Class Properties

In this section, we discuss each QueryRecord class property. The properties are discussed in alphabetical order.

Description

Description

This property returns the short description of the record as a string.

The length of this property is 30 characters.

This property is read-only.

JoinAlias

Description

This property returns or sets the join record's alias, that is, A, B, C, and so on.

This is applicable in any type of join. The value is a string. The length of this property is 1 character.

This property is read-write.

JoinFieldName

Description

This property returns or sets the join record's field used in the join criteria. This is applicable in lookup-table joins. The value is a string.

The length of this property is 18 characters.

This property is read-write.

JoinType

Description

This property returns or sets the join type. This is applicable in any type of join. You can use either a constant or numeric value.

The values for the join type are:

Numeric Value	***Constant Value***	***Description***
1	%Query_JoinNone	Query record isn't joined with any other record.

Numeric Value	***Constant Value***	***Description***
2	%Query_JoinHierarchy	Query record has a hierarchy join with another record.
3	%Query_JoinRelated	Query record has a lookup table join with another record.
4	%Query_JoinTree	Query record has a tree join with another record.
5	%Query_JoinLeftOuter	Query record has a left outer join with another record.

This property is read-write.

Name

Description

This property returns the name of the record as a string.

The length of this property is 15 characters.

This property is read-only.

QueryFields

Description

This property returns a reference to a QueryField Collection that contains instances of all the fields belonging to the record definition.

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryField Collection, page 1980

RecordAlias

Description

This property returns the alias used for the record in the query (that is, A, B, C, and so on.)

The length of this property is 1 character.

This property is read-write.

QueryField Collection

A QueryField collection is returned from the following:

- The QueryOutputFields of QuerySelect class method.
- The QuerySelectedFields of QuerySelect class method.
- The QueryFields QueryRecord class method.

The collection of QueryOutputFields does *not* necessarily include all columns returned in query resultset. This collection only includes columns that have been added to query output collection using the query classes or by an end-user designing a query.

PeopleSoft Query can also add additional columns for related language processing and also for translate labels on fields, and so you cannot use the Query Output Collections as a way to discover all of the columns that are returned in the resultset or by executing the SQL generated from a query.

See Chapter 33, "Query Classes," QueryOutputFields, page 1971; Chapter 33, "Query Classes," QuerySelectedFields, page 1893 and Chapter 33, "Query Classes," QueryFields, page 1979.

QueryField Collection Methods

In this section, we discuss each QueryField method. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first QueryField object in the QueryField collection.

Parameters

None.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&MyQueryField = &MyCollection.First();
```

Item

Syntax

Item*(number)*

Description

The Item method returns the QueryField object that exists at the *number* position in the QueryField collection.

Parameters

Parameter	*Description*
Number	Specify the position number in the collection of the QueryField object that you want returned.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryFieldColl.Count;
   &MyQueryField = &QueryFieldColl.Item(&I);
   /* do processing */
End-For;
```

ItemByNameAndAlias

Syntax

ItemByNameAndAlias*(Name, RecordAlias)*

Description

The ItemByNameAndAlias method returns the QueryField object with the given *Name* and *Record Alias.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing QueryField within the QueryField collection. If you specify an invalid name, the object is NULL.
RecordAlias	Alias of the record to which the field belongs

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&MyQueryField = &MyCollection.ItemByNameAndAlias("PHONELIST", "A");
```

ItemByExpNum

Syntax

`ItemByExpNum`*`(ExpNum)`*

Description

The ItemByExpNum method returns the QueryField object with the given Expression Number, *ExpNum*.

Parameters

Parameter	*Description*
ExpNum	Expression Number for the given expression field. Non-expression fields have 0 value for the Expression Number.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&QryFld = &QryFldCol.ItemByExpNum(1);
```


Next

Syntax

Next()

Description

The Next method returns the next QueryField object in the QueryField collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&MyQueryField = &MyCollection.Next();
```

QueryField Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryField objects in the QueryField Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryField Class

A QueryField object is returned by the following:

- The AddQuerySelectedField of QuerySelect class method.
- The AddQueryOutputField of QuerySelect class method.
- The QueryField collection methods First, Item, ItemByName or Next.

See Chapter 33, "Query Classes," AddQuerySelectedField, page 1965 and Chapter 33, "Query Classes," AddQueryOutputField, page 1964.

See Chapter 33, "Query Classes," QueryField Collection, page 1980.

QueryField Class Methods

In this section, we discuss each QueryField method. The methods are discussed in alphabetical order.

AddTranslateExpression

Syntax

`AddTranslateExpression`*`(ExpressionName)`*

Description

Use the AddTranslateExpression method to add Translate Effective Date Logic expressions for translatable fields.

Parameters

Parameter	***Description***
ExpressionName	Specify the name of the expression name that you want to add.

Returns

A reference to a QueryExpression object.

See Also

Chapter 33, "Query Classes," Expressions, page 1970; Chapter 33, "Query Classes," DeleteExpression, page 1967; Chapter 33, "Query Classes," AddExpression, page 1963; Chapter 33, "Query Classes," AddTranslateField, page 1985 and Chapter 33, "Query Classes," QueryExpression Class, page 2018

AddTranslateField

Syntax

`AddTranslateField``(`*`FieldName`*`)`

Description

Use the AddTranslateField method to add a Translate Effective Date Logic field for a translatable field.

Parameters

Parameter	***Description***
FieldName	Specify the name of the translate field that you want to add. The maximum allowed length of this parameter is 18 characters.

Returns

A reference to a QueryField object.

See Also

Chapter 33, "Query Classes," Expressions, page 1970; Chapter 33, "Query Classes," DeleteExpression, page 1967; Chapter 33, "Query Classes," AddExpression, page 1963; Chapter 33, "Query Classes," AddTranslateExpression, page 1984 and Chapter 33, "Query Classes," QueryExpression Class, page 2018

GetImageFormat

Syntax

`GetImageFormat``()`

Description

Use the GetImageFormat method to differentiate between images and files, both stored as binary large objects (BLOBs) in the database.

Note. Because images and files share the field type value 8, this method is required to differentiate between the two types.

Parameters

None.

Returns

A number. A value 16 indicates that the field is of type file (or attachment).

See Also

Chapter 33, "Query Classes," Type, page 1994

QueryField Class Properties

In this section, we discuss the QueryField class properties. The properties are discussed in alphabetical order.

Aggregate

Description

This property returns or sets the aggregate type for the query field.

This property takes either a constant or numeric value. The values are:

Numeric Value	*Constant Value*	*Description*
1	%Query_TtlNone	Query field has no total
2	%Query_TtlSum	Query field is for Sum
3	%Query_TtlCount	Query field is for Count
4	%Query_TtlMin	Query field is for Minimum
5	%Query_TtlMax	Query field is for Maximum
6	%Query_TtlAvg	Query field is for Average

This property is read-write.

ColumnNumber

Description

This property returns or sets the column number for the query field as a number. If the column number is greater than zero, the field is an output field.

This property is read-write.

Description

Description

This property returns the long description of the field as a string. This same value is returned by the Description property of the QueryDBRecordField class.

The length of this property is 30 characters.

This property is read-only.

Decimal

Description

This property returns the decimal positions QueryField as a number. This same value is returned by the Decimal property of the QueryDBRecordField class.

This property is read-only.

ExpNum

Description

This property returns or sets the Expression Number for the field. This value is 0 for non-expression fields. If the field is created for an expression, this value is the same as the expression number of the corresponding expression. This property takes a number value.

This property is read-write.

Flag

Description

This property returns the Use Edit flag for the Query Field. It has the same values as returned by the Flag property of the QueryDBRecordField. This property takes a number value.

This property is read-only.

Format

Description

This property returns the field format for the query field. This property returns a string value. This property is useful for displaying the data type in a string format.

Values are:

Data Type	***Value Returned***
Character	"CHAR "
Long Character	"LONG CHAR "
Number	"NUMBER "
Signed number	"SIGNED "
Date	"DATE "
Time	"TIME "
Datetime	"DATE/TIME "
Image	"IMAGE "

This property is read-only.

HeadingText

Description

This property returns or sets the heading text for the query field as a string.

The length of this property is 30 characters.

This property is read-write.

HeadingType

Description

This property returns or sets the heading type for the query field. This property takes either a numeric or constant value. The values are:

Numeric Value	***Constant Value***	***Description***
1	%Query_HdgNone	Query field has no heading.
2	%Query_HdgText	Query field has a text heading.
3	%Query_HdgRftShort	Query field uses the short RFT heading.
4	%Query_HdgRftLong	Query fields uses the long RFT heading.

This property is read-write.

HeadingUniqueFieldName

Description

This property returns or sets the unique field name for the heading. The default value for this property is the record alias combined with the field name.

The length of this property is 30 characters.

This property is read-write.

Length

Description

This property returns the length of the Query Field. The same value is returned by the Length property of the QueryDBRecordField. This property takes a numeric value.

This property is read-only.

LongName

Description

This property returns the long name of the Query Field. The same value is returned by the Long Name property of the QueryDBRecordField. This property takes a string value.

The length of this property is 30 characters.

This property is read-only.

Name

Description

This property returns the name of the query field as a string.

The length of this property is 18 characters.

This property is read-only.

OrderByDirection

Description

This property returns or sets the order by direction. This property takes a numeric value. The constants are the ASCII codes for space (" ") and "D".

Values are:

Numeric Value	***Constant Value***	***Description***
32	Code (" ")	Ascending
68	Code("D")	Descending

This property is read-write.

Example

The following sets the order by direction to be ascending.

```
&QueryField.OrderByDirection = Code(" ");
```

The following sets the order by direction to be descending.

```
&QueryField.OrderByDirection = Code("D");
```

OrderByNumber

Description

Use this property to specify whether a field is used as part of an 'Order By' statement in the SQL. The number value of this property indicates which is the first field in the order by statement, which is the second, and so on.

This property is read-write.

Example

To order a query by one field, you must set the OrderByNumber property for the other QueryFields as well, specifying the primary field as 1, the next field as 2, and so on.

```
&QryFld = &MainQrySel.AddQuerySelectedField("PSRECDEFN", "A", "RECNAME", ⇒
"Record Name");
&QryFld.ColumnNumber = 1;
&QryFld.OrderByNumber = 1;

&QryFld = &MainQrySel.AddQuerySelectedField("PSRECDEFN", "A", "RECDESCR", ⇒
"Record Descr");
&QryFld.ColumnNumber = 2;
&QryFld.OrderByNumber = 2;

&QryFld = &MainQrySel.AddQuerySelectedField("PSRECDEFN", "A", "RELLANGRECNAME",⇒
 "Record Lang Rec");
&QryFld.ColumnNumber = 3;
&QryFld.OrderByNumber = 3;
```

QueryRecord

Description

This property returns a reference to the QueryRecord containing this QueryField. If the field has no QueryRecord (that is, that this field is an expression field), this property returns NULL.

This property is read-only.

Example

```
&QryRcd = &QryFld.QueryRecord;
```

RecordAlias

Description

This property returns the record alias for the Query Field as a string. This value is usually a value like "A", "B", and so on. The same value is returned by the RecordAlias property for QueryRecord.

The length of this property is 1 character.

This property is read-only.

ShortName

Description

This property returns the short name of the Query Field. The same value is returned by the ShortName property for QueryDBRecordField. This property takes a string value.

The length of this property is 15 characters.

This property is read-only.

TranslateEffDtLogic

Description

This property returns or sets the effective date logic for the QueryField.

The values are:

Numeric Value	***Constant Value***	***Description***
1	%Query_ExprCurDt	Current Date
2	%Query_ExprField	Field
3	%Query_Expression	Expression

This property is read-write.

TranslateExpression

Description

This property returns a reference to an expression object based on a translate field if the Translate Effective Date option refers to an Expression

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryExpression Class, page 2018

TranslateField

Description

This property returns a reference to a QueryField object for a translate field if the Translate Effective Date Option refers to a field.

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryField Class, page 1984

TranslateOption

Description

This property returns or sets the translate value for the QueryField. This property takes a numeric or constant value. The values are:

Numeric Value	*Constant Value*	*Description*
1	%Query_XlatNone	None
2	%Query_XlatShort	Short
3	%Query_XlatLong	Long

This property is read-write.

Type

Description

This property returns the type of the field. You can use either a numeric or constant value. The values are:

Numeric Value	***Constant Value***	***Description***
0	%FieldType_Char	Character
1	%FieldType_LongChar	Long Character
2	%FieldType_Number	Number
3	%FieldType_SignedNumber	Signed number
4	%FieldType_Date	Date
5	%FieldType_Time	Time
6	%FieldType_DateTime	DateTime
8	%FieldType_Image	Image
8	%FieldType_File	File
9	%FieldType_ImageRef	ImageReference

Note. The Image and File types can be differentiated using the GetImageFormat method of the QueryField class.

This property is read-only.

See Also

Chapter 33, "Query Classes," GetImageFormat, page 1985

QueryCriteria Collection

A QueryCriteria collection is returned from the Criteria QuerySelect class property.

See Chapter 33, "Query Classes," Criteria, page 1970.

QueryCriteria Collection Methods

In this section, we discuss the QueryCriteria collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first QueryCriteria object in the QueryCriteria collection.

Parameters

None.

Returns

A reference to a QueryCriteria object if successful, NULL otherwise.

Example

```
&MyQueryCriteria = &MyCollection.First();
```

Item

Syntax

Item*(number)*

Description

The Item method returns the QueryCriteria object that exists at the *number* position in the QueryCriteria collection.

Parameters

Parameter	*Description*
Number	Specify the position number in the collection of the QueryCriteria object that you want returned.

Returns

A reference to a QueryCriteria object if successful, NULL otherwise.

Example

```
For &I = 1 to &MyCollection.Count;
   &MyQueryCriteria = &MyCollection.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

`ItemByName`*`(CriteriaName)`*

Description

The ItemByName method returns the QueryCriteria object that exists with the passed CriteriaName in the QueryCriteria collection.

Parameters

Parameter	*Description*
CriteriaName	Specify the name of the Criteria to be searched. This name is the same as the one used while creating the criteria using QuerySelect.AddCriteria(CriteriaName).

Returns

A reference to a QueryCriteria object if successful, NULL otherwise.

Example

```
&MyQueryCriteria = &MyCollection.ItemByName("PHONELIST");
```


Next

Syntax

Next()

Description

The Next method returns the next QueryCriteria object in the QueryCriteria collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryCriteria object if successful, NULL otherwise.

Example

```
&MyQueryCriteria = &MyCollection.Next();
```

QueryCriteria Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryCriteria objects in the QueryCriteria Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryCriteria Class

A QueryCriteria object is returned from the following:

- The AddCriteria and AddHavingCriteria QuerySelect class method.
- The First, Item, and Next method of the QueryCriteria Collection.

See Chapter 33, "Query Classes," AddCriteria, page 1962 and Chapter 33, "Query Classes," AddHavingCriteria, page 1963.

See Chapter 33, "Query Classes," QueryCriteria Collection, page 1994.

See Chapter 33, "Query Classes," Working With Query Criteria and Expressions, page 1894.

See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query*, "Defining Selection Criteria."

QueryCriteria Class Methods

In this section, we discuss the QueryCriteria class methods. The methods are discussed in alphabetical order.

AddExpr1Expression

Syntax

`AddExpr1Expression``()`

Description

The AddExpr1Expression method returns a reference to a new a QueryExpression object to be used as an expression for Expression 1. You can then use this object to specify details about the expression using the methods and properties of the QueryExpression Class.

Note. You must set the type for Expression 1 using the Expr1Type property before you can use this method.

Parameters

None.

Returns

A reference to a blank QueryExpression object.

See Also

Chapter 33, "Query Classes," Expr1Expression, page 2003; Chapter 33, "Query Classes," AddExpr1Field, page 1999; Chapter 33, "Query Classes," Expr1Type, page 2004 and Chapter 33, "Query Classes," QueryExpression Class, page 2018

AddExpr1Field

Syntax

`AddExpr1Field``(`*`QueryRecordAlias,FieldName`*`)`

Description

The AddExpr1Field method returns a reference to a new a QueryField object to be used as a field for Expression 1. You can then use this object to specify details about the expression using the methods and properties of the QueryField Class.

Note. You must set the type for Expression 1 using the Expr1Type property before you can use this method.

Parameters

Parameter	***Description***
QueryRecordAlias	Specify the alias of the QueryRecord that contains the QueryField that you want to use.
FieldName	Specify the name of the QueryField in the QueryRecord that you want to use.

Returns

A reference to a blank QueryField object.

See Also

Chapter 33, "Query Classes," Expr1Field, page 2003; Chapter 33, "Query Classes," AddExpr1Expression, page 1998; Chapter 33, "Query Classes," Expr1Type, page 2004 and Chapter 33, "Query Classes," QueryField Class, page 1984

AddExpr2Expression

Syntax

`AddExpr2Expression()`

Description

The AddExpr2Expression method returns a reference to a new a QueryExpression object to be used as an expression for Expression 2. You can then use this object to specify details about the expression using the methods and properties of the QueryExpression Class.

Parameters

None.

Returns

A reference to a blank QueryExpression object.

See Also

Chapter 33, "Query Classes," QueryExpression Class, page 2018

AddExpr2Field1

Syntax

AddExpr2Field1*(QueryRecordAlias,FieldName)*

Description

Expression 2 has two field expressions because when the Between relational operator is used in a criteria, there can be two expression fields. The AddExpr2Field1 method returns a reference to a new QueryField object to be used as a *field 1* for Expression 2. You can then use this object to specify details about the expression using the methods and properties of the QueryField Class.

Parameters

Parameter	*Description*
QueryRecordAlias	Specify the alias of the QueryRecord that contains the QueryField that you want to use.
FieldName	Specify the name of the QueryField in the QueryRecord that you want to use.

Returns

A reference to a blank QueryField object.

See Also

Chapter 33, "Query Classes," QueryRecord Class, page 1976 and Chapter 33, "Query Classes," QueryField Class, page 1984

AddExpr2Field2

Syntax

`AddExpr2Field2`*`(QueryRecordAlias,FieldName)`*

Description

Expression 2 has two field expressions because when the Between relational operator is used in a criteria, there can be two expression fields. The AddExpr2Field2 method returns a reference to a new a QueryField object to be used as a *field 2* for Expression 2. You can then use this object to specify details about the expression using the methods and properties of the QueryField Class.

Parameters

Parameter	*Description*
QueryRecordAlias	Specify the alias of the QueryRecord that contains the QueryField that you want to use.
FieldName	Specify the name of the QueryField in the QueryRecord that you want to use.

Returns

A reference to a blank QueryField object.

See Also

Chapter 33, "Query Classes," QueryRecord Class, page 1976 and Chapter 33, "Query Classes," QueryField Class, page 1984

AddExpr2List

Syntax

AddExpr2List()

Description

Expression 2 can have a list of values when the Operator is of type In List (or Not In List). The AddExpr2List method returns a reference to a new QueryList, which can be used to add a list of values to the criteria.

Parameters

None.

Returns

A reference to a blank QueryList object.

See Also

Chapter 33, "Query Classes," QueryList Class, page 2023

AddExpr2Subquery

Syntax

AddExpr2Subquery()

Description

The AddExpr2Subquery method is used to create a subquery for Expression2. This method returns a new QuerySelect object you can use to specify details about the new subquery.

Warning! The new subquery created with this method replaces any existing subquery (for this criteria), destroying any existing properties or values.

Parameters

None.

Returns

A reference to a new QuerySelect object.

See Also

Chapter 33, "Query Classes," QuerySelect Class, page 1960

QueryCriteria Class Properties

In this section, we discuss the QueryCriteria class properties. The properties are discussed in alphabetical order.

Expr1Expression

Description

This property returns a reference to the QueryExpression object that's used as Expression 1.

This property is valid only when Expression 1 exists as an expression. If you want to add an expression for Expression 1, use the AddExpr1Expression method.

This property is read-write.

See Also

Chapter 33, "Query Classes," AddExpr1Expression, page 1998

Expr1Field

Description

This property returns a reference to the QueryField object that's used as Expression 1.

This property is valid only when Expression 1 exists as a field. You can then use the QueryField Class methods and property to manipulate this object.

If you want to add a field for Expression 1, use the AddExpr1Field method.

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryField Class, page 1984 and Chapter 33, "Query Classes," AddExpr1Field, page 1999

Expr1Type

Description

This property returns or sets the type for Expression 1.

Note. You *must* set the type of expression for every new criteria.

The values are:

Numeric Value	***Constant Value***	***Description***
2	%Query_ExprField	Field
3	%Query_Expression	Expression

This property is read-write.

Example

The following is used to test the expression to determine the property to use to retrieve it:

```
&MyExpr1 = &MyCrtColl.Next();
If &MyExpr1.Expr1Type = %Query_ExprField Then /* Expression is a Field */

   &OldExpr1Value = &MyExpr1.Expr1Field;
   /* do processing */

Else  /* Expression 1 is an expression */

   &OldExpr1Value = &MyExpr1.Expr1Expression;
   /* Do processing */

End-if;
```

The following is an example showing how to add a field for Expression 1.

```
/* add a new criteria */
&MyCriteria = &MyQuery.AddCriteria();

/* set the type of the first expression to be a field */
&MyCriteria.Expr1Type = %Query_ExprField;

/* add the field EMPLID from the ABSENCE_HIST record */
&MyField = &MyCriteria.AddExpr1Field("A", "EMPLID");
```


Expr2Constant1

Description

If the Between relational operator is used in the criteria, there can be two constants for Expression 2. This property returns or sets the constant value for the first constant for Expression 2. This property takes a string value.

This property is valid only when Expression 2 is defined as a constant.

This property is read-write.

Expr2Constant2

Description

If the Between relational operator is used in the criteria, there can be two constants for Expression 2. This property returns or sets the constant value for the second constant for Expression 2. This property takes a string value.

This property is valid only when Expression 2 is defined as a constant.

This property is read-write.

Expr2Expression1

Description

If the Between relational operator is used in the criteria, there can be two expressions for Expression 2. This property returns a reference to the first QueryExpression object that's used as Expression 2.

This property is valid only when Expression 2 exists as an expression. To add an expression for Expression 2, use the AddExpr2Expression method.

This property is read-write.

See Also

Chapter 33, "Query Classes," AddExpr2Expression, page 2000

Expr2Expression2

Description

If the Between relational operator is used in the criteria, there can be two expressions for Expression 2. This property returns a reference to the second QueryExpression object that's used as Expression 2.

This property is valid only when Expression 2 exists as an expression. To add an expression for Expression 2, use the AddExpr2Expression method.

This property is read-write.

See Also

Chapter 33, "Query Classes," AddExpr2Expression, page 2000

Expr2Field1

Description

If the Between relational operator is used in the criteria, there can be two fields for Expression 2. This property returns a reference to the first QueryField object that's used as Expression 2.

This property is only valid when Expression 2 exists as a field. You can then use the QueryField Class methods and property to manipulate this object.

To add a field for Expression 2, use the AddExpr2Field1 method.

This property is read-only.

See Also

Chapter 33, "Query Classes," AddExpr2Field1, page 2000 and Chapter 33, "Query Classes," QueryField Class, page 1984

Expr2Field2

Description

If the Between relational operator is used in the criteria, there can be two fields for Expression 2. This property returns a reference to the second QueryField object that's used as Expression 2.

This property is valid only when Expression 2 exists as a field. You can then use the QueryField Class methods and property to manipulate this object.

To add a field for Expression 2, use the AddExpr2Field2 method.

This property is read-only.

See Also

Chapter 33, "Query Classes," AddExpr2Field2, page 2001 and Chapter 33, "Query Classes," QueryField Class, page 1984

Expr2List

Description

This property returns a reference to the QueryList object of Expression 2 that's used when the Operator is of type In List (or Not In List).

This property is read-only.

Expr2Subquery

Description

This property returns a reference to the QuerySelect object that's used as a subquery for Expression 2.

This property is valid only when Expression 2 exists as a subquery. To add a subquery for Expression 2, use the AddExpr2Subquery method.

This property is read-only.

See Also

Chapter 33, "Query Classes," AddExpr2Subquery, page 2002

Expr2Type

Description

This property returns or sets the type for Expression 2. The following table lists all of possible values for this property. However, the values for this property are dependent upon the Operator property.

This property is read-write.

See Chapter 33, "Query Classes," Operator, page 2012 and Chapter 33, "Query Classes," Working With Query Criteria and Expressions, page 1894.

You can use either a constant or numeric value for this property. The values are:

Numeric Value	***Constant Value***	***Description***
1	%Query_ExprConstant	Constant
2	%Query_ExprField	Field
3	%Query_Exprression	Expression
4	%Query_ExprSubQuery	Subquery
5	%Query_ExprList	List
6	%Query_ExprCurDt	Current date
7	%Query_ExprTree	Tree
8	%Query_ExprBind	Bind
9	%Query_ExprBothConst	The criterion's operator is Between and both values on the right-hand side are constants. (Const-Const)
10	%Query_ExprConstFld	The criterion's operator is Between, the first value on right-hand side is a constant and the second value is a field. (Const-Field)
11	%Query_ExprConstExpr	The criterion's operator is Between, the first value on right-hand side is a constant and the second value is an expression. (Const-Expr)
12	%Query_ExprFieldConst	The criterion's operator is Between, the first value on right-hand side is a field and the second value is a constant. (Field-Const)
13	%Query_ExprBothFld	The criterion's operator is Between and both values on the right-hand side are constants. (Field-Field)
14	%Query_ExprFldExpr	The criterion's operator is Between, the first value on right-hand side is a field and the second value is an expression. (Field-Expr)
15	%Query_ExprExprConst	The criterion's operator is Between, the first value on right-hand side is an expression and the second value is a constant. (Expr-Const)

Numeric Value	Constant Value	Description
16	%Query_ExprExprFld	The criterion's operator is Between, the first value on right-hand side is an expression and the second value is a field. (Expr-Field)
17	%Query_ExprBothExpr	The criterion's operator is Between and both values on the right-hand side are expressions. (Expr-Expr)
18	%Query_ExprTreePrompt	Tree prompt

The following table describes how to access or change Expression 2 depending on the Expression2 Type.

Expression2 Type	Method or Property for Changing the Expression	Method or Property for Accessing the Expression
Constant	Expr2Constant	Expr2Constant
Field	AddExpr2Field()	Expr2Field
Expression	AddExpr2Expression()	Expr2Expression
In List	AddExpr2List	Expr2List
In Tree	AddExpr2Expression	Expr2Expression
Subquery	AddExpr2Subquery()	Expr2Subquery
Const-Const	Expr2Constant, Expr2Constant	Expr2Constant, Expr2Constant
Const-Field	Expr2Constant, AddExpr2Field()	Expr2Constant, Expr2Field
Const-Expr	Expr2Constant, AddExpr2Expression()	Expr2Constant, Expr2Expression
Field-Const	AddExpr2Field(), Expr2Constant	Expr2Field, Expr2Constant

Expression2 Type	*Method or Property for Changing the Expression*	*Method or Property for Accessing the Expression*
Field-Field	AddExpr2Field(), AddExpr2Field()	Expr2Field, Expr2Field
Field-Expr	AddExpr2Field(), AddExpr2Expression()	Expr2Field, Expr2Expression
Expr-Const	AddExpr2Expression(), Expr2Constant	Expr2Expression, Expr2Constant
Expr-Field	AddExpr2Expression(), AddExpr2Field()	Expr2Expression, Expr2Field
Expr-Expr	AddExpr2Expression(), AddExpr2Expression()	Expr2Expression, Expr2Expression

Example

The following is used to test the expression to determine the property to use to retrieve it:

```
&MyExpr2 = &MyCriteria.Expr2Expression;

If &MyExpr2.Expr2Type = %Query_ExprConstant Then /* Expression is a constant */

   &OldValue = &MyExpr2.Expr2Constant;
   /* do processing */

End-if;
```

The following is an example showing how to add a field for Expression 2:

```
( )
/* After setting the first expression, set the operator for the criteria*/
&MyCriteria.Operator = %Query_CondEqual;

/* Set the type of the second expression to be a field */
&MyCriteria.Expr2Type = %Query_ExprField;

/* Add the EMPLID field from the ABSENCE_HIST record whose record alias is A */
&MyField = &MyCriteria.AddExpr2Field("A", "EMPLID" );
```

Logical

Description

This property returns or sets the logical portion of a criteria.

Note. This property is valid only when there are more than one criteria for a query. Also, this property is required when there is more than one criteria for a query.

The values are:

Numeric Value	***Constant Value***	***Description***
1	%Query_CombAnd	Logical And for non-having criteria
2	%Query_CombOr	Or for non-having criteria
3	%Query_CombNotUsed	No logical operator. Used for the first non-having criteria.
4	%Query_CombHaveAnd	Logical And for having criteria
5	%Query_CombHaveOr	Logical Or for having Criteria
6	%Query_CombHaveNotUsed	No logical operator. Used for the first having criteria.

This property is read-write.

LParenLvl

Description

This property returns or sets the left parenthesis level used for grouping criteria. This property takes a numeric value.

This property is read-write.

Name

Description

This property returns the name of the Query Criteria, as a string.

This property is read-only.

Negation

Description

This property returns a Boolean value, indicating whether a criterion is negated: True if the criterion is negated, False if it isn't.

This property is read-write.

Operator

Description

This property returns or sets the operator for the criteria.

The value of this property determines the valid types of the Expression 2, set with the Expr2Type property.

This property is read-write.

See Chapter 33, "Query Classes," Expr2Type, page 2007 and Chapter 33, "Query Classes," Working With Query Criteria and Expressions, page 1894.

You can use either a constant or numeric value for this property. The values are:

Numeric Value	***Constant Value***	***Description***
1	%Query_CondNone	None (used for initializing a new criteria.)
2	%Query_CondEqual	Criteria's left-hand side is equal to right-hand side (= operator)
3	%Query_CondNotEqual	Criteria's left-hand side is not equal to right-hand side (<> operator)
4	%Query_CondGreaterThan	Criteria's left-hand side is greater than right-hand side (> operator)
5	%Query_CondNotGreaterThan	Criteria's left-hand side is not greater than right-hand side (<= operator)
6	%Query_CondLessThan	Criteria's left-hand side is less than right-hand side (< operator)
7	%Query_CondNotLessThan	Criteria's left-hand side is not less than right-hand side (>= operator)
8	%Query_CondInList	Criteria's left-hand side is in the given list (IN operator)

Numeric Value	***Constant Value***	***Description***
9	%Query_CondNotInList	Criteria's left-hand side is not in the given list (Not IN operator)
10	%Query_CondBetween	Criteria's left-hand side is between the two values of right-hand side (BETWEEN operator)
11	%Query_CondNotBetween	Criteria's left-hand side is not between the two values of right-hand side (BETWEEN operator)
12	%Query_CondExists	Criteria's left-hand side is the output of the subquery of right-hand side (EXISTS operator)
13	%Query_CondNotExists	Criteria's left-hand side doesn't exist in the output of the subquery of right-hand side (NOT EXISTS operator)
14	%Query_CondLike	Criteria's left-hand side is like (wildcard search) the right-hand side (LIKE operation)
15	%Query_CondNotLike	Criteria's left-hand side is not like (wildcard search) the right-hand side (NOT LIKE operation)
16	%Query_CondNull	Criteria's left-hand side is NULL (NULL operation)
17	%Query_CondNotNull	Criteria's left-hand side is not NULL (IS NOT NULL operation)
18	%Query_CondInTree	Criteria's left-hand side is from a list of nodes in Tree (IN operation)
19	%Query_CondNotInTree	Criteria's left-hand side is not from a list of nodes in Tree (NOT IN operation)
20	%Query_CondEffDtLessEqual	Criteria's left-hand side is an Effective Date and is less than or equal to the date on the right-hand side (<= operation)
21	%Query_CondEffDtGreaterEqual	Criteria's left-hand side is an Effective Date and is greater than or equal to the date on the right-hand side (>= operation)

Numeric Value	Constant Value	Description
22	%Query_CondEffDtLess	Criteria's left-hand side is an Effective Date and is less than the date on the right-hand side (< operation)
23	%Query_CondEffDtGreater	Criteria's left-hand side is an Effective Date and is greater than the date on the right-hand side (> operation)
24	%Query_CondFirstEffDt	Criteria's left-hand side is the first effective date (Function MIN())
25	%Query_CondLastEffDt	Criteria's left-hand side is the last effective date (Function MAX())
26	%Query_CondInTreeJoin	Criteria's left-hand side is an In Tree Join.

R1ExprNum

Description

This property returns or sets the expression number for the first expression of Expression 2. This property takes a numeric value.

This property is read-write.

R2ExprNum

Description

This property returns or sets the expression number for the second expression of Expression 2. This property takes a numeric value.

This property is read-write.

R1ExprType

Description

This property returns the expression type for the first expression of Expression 2. This property takes a numeric value and is the same range of values as the Expr2Type. It helps distinguish the type of an expression based on the value of Expr2Type. For instance, if the Expr2Type is Field-Expr, R1Expr2Type is of type Field and R2Expr2Type is of type Expression.

This property is read-only.

R2ExprType

Description

This property returns the expression type for the second expression of Expression 2. This property takes a numeric value and is the same range of values as the Expr2Type. It helps distinguish the type of an expression based on the value of Expr2Type. For instance, if the Expr2Type is Field-Expr, R1Expr2Type is of type Field and R2Expr2Type is of type Expression.

This property is read-only.

RParenLvl

Description

This property returns or sets the right parenthesis level used for grouping criteria. This property takes a numeric value.

This property is read-write.

QueryExpression Collection

A QueryExpression Collection is returned from the Expressions Query class property.

See Chapter 33, "Query Classes," Expressions, page 1970.

QueryExpression Collection Methods

In this section, we discuss the QueryExpression collection methods. The methods are described in alphabetical order.

First

Syntax

First()

Description

The First method returns the first QueryExpression object in the QueryExpression collection.

Parameters

None.

Returns

A reference to a QueryExpression object if successful, NULL otherwise.

Example

```
&MyQueryExpression = &MyCollection.First();
```

Item

Syntax

`Item`*`(number)`*

Description

The Item method returns the QueryExpression object that exists at the *number* position in the QueryExpression collection.

Parameters

Parameter	***Description***
Number	Specify the position number in the collection of the QueryExpression object that you want returned.

Returns

A reference to a QueryExpression object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryExpressionColl.Count;
   &MyQueryExpression = &QueryExpressionColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName*(ExpressionName)*

Description

The ItemByName method returns the QueryExpression object in the QueryExpressionCollection with the given expression name.

Parameters

Parameter	*Description*
ExpressionName	The name of the required expression.

Returns

A reference to a QueryExpression object if successful, NULL otherwise.

Example

```
&QryExpr = &QryExprCol.Item("Exp-6");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryExpression object in the QueryExpression collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryExpression object if successful, NULL otherwise.

Example

```
&MyQueryExpression = &MyCollection.Next();
```

QueryExpression Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryExpression objects in the QueryExpression Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryExpression Class

A QueryExpression object is returned by the following:

- The AddExpression method of the Query class.
- The First, Item and Next methods of the QueryExpression collection.
- Some of the QueryCriteria class methods.
- The Expr1Expression and Expr2Expression1 properties of the QueryCriteria class.

See Chapter 33, "Query Classes," AddExpression, page 1963; Chapter 33, "Query Classes," QueryExpression Collection, page 2015; Chapter 33, "Query Classes," QueryCriteria Class Methods, page 1998; Chapter 33, "Query Classes," Expr1Expression, page 2003 and Chapter 33, "Query Classes," Expr2Expression1, page 2005.

See Chapter 33, "Query Classes," Working With Query Criteria and Expressions, page 1894.

QueryExpression Class Properties

In this section, we discuss the QueryExpression class properties. The properties are discussed in alphabetical order.

Aggregate

Description

This property specifies whether the expression is an aggregate function.

This property takes a Boolean value: True if the expression is an aggregate function, False, otherwise.

This property is read-write.

BindFlag

Description

This property specifies whether the expression contains a bind value, as Boolean value.

This property is mainly used while reading a query, to determine whether a Query Expression contains a bind value. It isn't necessary to set this value while saving a query to the database.

Values are:

Value	***Description***
False	Expression doesn't have a bind value
True	Expression has a bind value

This property is read-write.

Decimal

Description

This property returns or sets the decimal value of the expression.

This property is only valid with numeric fields.

This property is read-write.

ExpNum

Description

This property returns or sets the unique expression number, as a numeric value.

This property is read-write.

IsXlatExpression

Description

This property indicates whether the expression is for a translate field. This property takes a Boolean value: True, the expression is based on a translated field.

This property is read-only.

Length

Description

This property returns or sets the length of the expression, as a number.

This property is read-write.

Name

Description

This property returns the name of the expression, as a string.

This property is read-only.

OutputField

Description

This property returns the instance of the displayed expression field, as a QueryField. This property returns a NULL when the expression isn't used in a displayed (output) field.

This property is read-write.

RightExprFlag

Description

This property specifies whether an expression is used in the right-hand side of a criteria (that is, is it an Expr2 expression), as a Boolean value.

This property is typically used while reading a query to determine whether a Query Expression is used in the right-hand side of criteria. It's not necessary to set this value while saving a query to the database.

Values are:

Value	***Description***
False	Expression is not used in the right-hand side of the criteria
True	Expression is used in the right-hand side of the criteria

This property is read-write.

SelectedField

Description

This property returns the instance of the expression field, as a QueryField. This property returns a NULL when the expression is used only in the right-hand side of a criteria.

This property is read-write.

Text

Description

This property returns or sets the text of the expression, as a character string.

This property is read-write.

Using ORACLE Hints in Expressions

Oracle hints can be included in expressions using the following considerations:

- Expression containing a hint must begin with /*+.
- Expression can only contain one hint. For example, only one set of /* */ is allowed in each expression.
- Each /* must precede an */.
- Each expression must contain a complete hint. For example, an expression can't have only /* or */. Both must be in same expression.

Type

Description

This property returns or sets the field type of the expression.

You can specify either a constant or a numeric value. The values are:

Numeric Value	***Constant Value***	***Description***
0	%FieldType_Char	Character
1	%FieldType_LongChar	Long Character
2	%FieldType_Number	Number
3	%FieldType_SignedNumber	Signed number
4	%FieldType_Date	Date
5	%FieldType_Time	Time
6	%FieldType_DateTime	DateTime
7	%FieldType_Image	Image
11	%FieldType_URL	Drilling URL

This property is read-write.

QueryList Class

A QueryList Class is returned from the Expr2List property and AddExpr2List method of the QueryCriteria Class.

See Chapter 33, "Query Classes," Expr2List, page 2007 and Chapter 33, "Query Classes," AddExpr2List, page 2002.

QueryList Class Methods

In this section, we discuss the QueryList class methods. The methods are discussed in alphabetical order.

AddListValue

Syntax

AddListValue*(Value,IsPrompt)*

Description

The AddListValue method adds a new List Value into the QueryList instance.

Parameters

Parameter	***Description***
Value	Specify the string value to be added to the list
IsPrompt	Specify whether the string specified by *Value* is a bind variable. This parameter takes a Boolean value: True, *Value* is a bind variable.

Returns

A reference to a QueryListValue object if successful, NULL otherwise.

Example

```
&MyListValue = &MyList.AddListValue("1", False);
```

First

Syntax

`First``()`

Description

The First method returns the first QueryListValue object in the QueryList instance.

Parameters

None.

Returns

A reference to a QueryListValue object if successful, NULL otherwise.

Example

```
&MyListValue = &MyList.First();
```

Item

Syntax

`Item`*`(number)`*

Description

The Item method returns the QueryListValue object that exists at the *number* position in the QueryList instance.

Parameters

Parameter	***Description***
Number	Specify the position number in the QueryList object that you want returned.

Returns

A reference to a QueryListValue object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryList.Count;
   &QryListVal = & QueryList.Item(&I);
   /* do processing */
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next QueryListValue object in the QueryList instance. This method should be called after calling First, else it returns NULL.

Parameters

None.

Returns

A reference to a QueryListValue object if successful, NULL otherwise.

Example

```
&MyNextListValue = &MyList.Next();
```

QueryList Class Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryListValue objects in the QueryList Collection, as a number.

This property is read-only.

Example

```
&ListValueCount = &MYList.Count;
```

QueryListValue Class

A QueryListValue instance is returned from the AddListValue, First, Item, or Next QueryList Class methods.

See Chapter 33, "Query Classes," QueryList Class Methods, page 2023.

QueryListValue Class Properties

In this section, we discuss the QueryListValue class properties. These properties are discussed in alphabetical order.

IsPrompt

Description

This property indicates whether the value is a bind variable (such as :1). This property returns a Boolean value: True, this property is a bind variable.

This property is read-only.

Value

Description

This property returns a string for the value stored in the list.

This property is read-only.

QueryRecordHierarchy Collection

A QueryRecordHierarchy collection is returned from the RecordHierarchy QueryDBRecord property.

The order of each QueryRecordHierarchy object in the collection maps to the order of each tree node as it appears in the tree hierarchy from top to bottom.

See Chapter 33, "Query Classes," RecordHierachy, page 2047.

QueryRecordHierarchy Collection Methods

In this section, we discuss the QueryRecordHierarchy collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first QueryRecordHierarchy object in the QueryRecordHierarchy collection.

Parameters

None.

Returns

A reference to a QueryRecordHierarchy object if successful, NULL otherwise.

Example

```
&MyQueryRecordHierarchy = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QueryRecordHierarchy object that exists at the *number* position in the QueryRecordHierarchy collection.

Parameters

Parameter	***Description***
Number	Specify the position number in the collection of the QueryRecordHierarchy object that you want returned.

Returns

A reference to a QueryRecordHierarchy object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryRecordHierarchyColl.Count;
   &MyQueryRecordHierarchy = &QueryRecordHierarchyColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the QueryRecordHierarchy object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing QueryRecordHierarchy within the QueryRecordHierarchy collection. If you specify an invalid name, the object is NULL.

Returns

A reference to a QueryRecordHierarchy object if successful, NULL otherwise.

Example

```
&MyQueryRecordHierarchy = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryRecordHierarchy object in the QueryRecordHierarchy collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryRecordHierarchy object if successful, NULL otherwise.

Example

```
&MyQueryRecordHierarchy = &MyCollection.Next();
```


QueryRecordHierarchy Collection Property

In this section, we discuss the QueryRecordHierarchy properties. These properties are discussed in alphabetical order.

Count

Description

This property returns the number of QueryRecordHierarchy objects in the QueryRecordHierarchy Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryRecordHierarchy Class

A reference to a QueryRecordHierarchy object is returned by the First, Item, ItemByName, and Next QueryRecordHierarchy Collection methods.

Every QueryRecordHierarchy object returned from a collection represents a record node in the Record Hierarchy tree in Query tab of PeopleSoft Query.

The QueryRecordHierarchy object returned from the QueryField represents the prompt table for the record field.

See Chapter 33, "Query Classes," QueryRecordHierarchy Collection, page 2027.

QueryRecordHierarchy Class Properties

In this section, we discuss the QueryRecordHierarchy class properties. The properties are discussed in alphabetical order.

Description

Description

This property returns a description of the record node as a string.

This property is read-only.

Level

Description

This property returns the level of the record node in the record hierarchy. 1 is the root node, 2 is a node beneath the root node, 3 is a child of that, and so on.

This property is read-only.

Name

Description

This property returns the name of the record node as a string.

This property is read-only.

ParentFlag

Description

This property returns the parent flag. The values are:

Value	***Description***
0	Record node contains no children nodes.
1	Record node contains children nodes

This property is read-only.

Query Metadata Collection

A Query Metadata collection is returned by the Metadata Query class property.

See Chapter 33, "Query Classes," Metadata, page 1950.

See Chapter 33, "Query Classes," Using Query Metadata, page 1898.

Query Metadata Collection Methods

In this section, we discuss the Query Metadata collection methods. These methods are discussed in alphabetical order.

First

Syntax

`First``()`

Description

The First method returns the first Query Metadata object in the Query Metadata collection.

Parameters

None.

Returns

A reference to a Query Metadata object if successful, NULL otherwise.

Example

```
&MyMetadata = &MyCollection.First();
```

Item

Syntax

`Item`*`(number)`*

Description

The Item method returns the Query Metadata object that exists at the *number* position in the Query Metadata collection.

Parameters

Parameter	*Description*
number	Specify the position number in the collection of the Query Metadata object that you want returned.

Returns

A reference to a Query Metadata object if successful, NULL otherwise.

Example

```
For &I = 1 to &MetadataColl.Count;
   &MyMetadata = &MetadataColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the Query Metadata object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing Query Metadata within the Query Metadata collection. If you specify an invalid name, the object is NULL.

Returns

A reference to a Query Metadata object if successful, NULL otherwise.

Example

```
&MyMetadata = &MyQuery.Metadata.ItemByName("Descr");
```


Next

Syntax

```
Next()
```

Description

The Next method returns the next Query Metadata object in the Query Metadata collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a Query Metadata object if successful, NULL otherwise.

Example

```
&MyMetadata = &MyCollection.Next();
```

Query Metadata Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Query Metadata objects in the Query Metadata Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```


Query Metadata Class

A Query Metadata object is returned from the Query Metadata Collection methods First, Item, ItemByName, or Next.

See Chapter 33, "Query Classes," Query Metadata Collection, page 2031.

See Chapter 33, "Query Classes," Using Query Metadata, page 1898.

Query Metadata Class Properties

In this section, we discuss the Query Metadata class properties. The properties are discussed in alphabetical order.

Name

Description

This property returns the name of each Query Metadata property as a string.

Values for this property are:

Value	***Description***
Descr	Description
LongDescr	Long description
Public/Private	Specifies if the query is public or private. If the query is private, the name of the owner is listed in Value.
LastUpdDttm	Last updated date and time
LastUpdOprId	The UserId of the user who updated the value last
Record	Record name. May be more than one.
Input Param	Input parameter. May be more than one.
Expression	Expression. May be more than one.
Field	Record field name for the output column. May be more than one.
Heading	Heading name for the output column. May be more than one.

This property is read-only.

Value

Description

This property returns the value for the Query Metadata property as a string.

This property is read-only.

QueryStatistics Class

The QueryStatistics class is used to view statistical information about a query's execution. It can be useful for query administration. You can view query statistics for a query before you save it to the database.

A QueryStatistics Class is returned from the QueryStatistics Query property.

See Chapter 33, "Query Classes," QueryStatistics, page 1954.

QueryStatistics Class Properties

In this section, we discuss the QueryStatistics properties. The properties are discussed in alphabetical order.

AvgExecTime

Description

This property returns the average execution time for the query as a number.

This property is read-only.

AvgFetchTime

Description

This property returns the average fetch time for the query as a number.

This property is read-only.

AvgNumRows

Description

This property returns the average number of rows fetched for the query.

This property is read-only.

ExecCount

Description

This property returns the total number of times the query has been executed.

This property is read-only.

LastExecDtTm

Description

This property returns the last date and time the query was executed, as a string.

This property is read-only.

QuerySecurityProfile Class

The QuerySecurityProfile class is used to view the current user's security profile for PeopleSoft Query. This class doesn't contain any methods, and all the properties are read-only. An instance of this class is returned by the GetQuerySecurityProfile Session method.

See Chapter 33, "Query Classes," GetQuerySecurityProfile, page 1913.

QuerySecurityProfile Class Properties

In this section, we discuss the QuerySecurityProfile properties. The properties are discussed in alphabetical order.

AllowAnyJoin

Description

This property indicates whether the user is allowed to define queries with any join. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

AllowDistinct

Description

This property indicates whether the user can define queries with a Distinct clause in a SELECT statement. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

AllowExpressions

Description

This property indicates whether the user is allowed to define expressions in queries. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

AllowSubqueries

Description

This property indicates whether the user is allowed to define criteria containing subqueries. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

AllowUnions

Description

This property indicates whether the user is allowed to define queries containing unions. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

ApprovePrivateQuery

Description

This property indicates whether the user is allowed to approve private queries. This property takes a Boolean value: True, the user can approve such queries. This property is meant for query administration.

This property is read-only.

ApprovePublicQuery

Description

This property indicates whether the user is allowed to approve public queries. This property takes a Boolean value: True, the user can approve such queries. This property is meant for query administration.

This property is read-only.

CanCreatePublic

Description

This property indicates whether the user can create public queries. This property takes a Boolean value: True, the user can create such queries.

This property is read-only.

CanCreateWorkFlow

Description

This property indicates whether the user can create or run any workflow queries. Workflow queries are of the following types:

- Archive
- Process
- Role

This property takes a Boolean value: True, the user can create such queries.

This property is read-only.

CanModifyQuery

Description

This property indicates whether the user can modify queries. This property takes a Boolean value: True, the user can modify queries.

This property is read-only.

CanRunQuery

Description

This property indicates whether the user can run queries. This property takes a Boolean value: True, the user can run queries.

This property is read-only.

CanRunToCrystal

Description

This property indicates whether the user can run queries to a Crystal report. This property takes a Boolean value: True, the user can run queries.

This property is read-only.

CanRunToExcel

Description

This property indicates whether the user can run queries to an Excel spreadsheet. This property takes a Boolean value: True, the user can run queries.

This property is read-only.

LimitUnapproved

Description

This property indicates whether there is a limit on the number of rows returned for unapproved queries. This property is meant for query administration. This property takes a Boolean value: True, limit the number of rows. The MaxUnapprovedRows property is active only if this property is specified as True.

This property is read-only.

See Also

Chapter 33, "Query Classes," MaxUnapprovedRows, page 2042

MaxInTreeCriteria

Description

This property indicates the maximum number of In Tree Criteria that can be used in the queries defined by the current user. This property takes a numeric value.

This property is read-only.

MaxJoins

Description

This property indicates the maximum number of joins allowed in the queries defined by the current user. This property takes a numeric value.

This property is read-only.

MaxRowsToFetch

Description

This property indicates the maximum number of rows to fetch for the current user when a query is executed. This property takes a numeric value.

This property is read-only.

MaxUnapprovedRows

Description

This property indicates the maximum number of rows that can be returned for unapproved queries. This property takes a numeric value. This property is meant for query administration. This property is active only if the property LimitUnapproved is True.

This property is read-only.

See Also

Chapter 33, "Query Classes," LimitUnapproved, page 2041

QueryDBRecord Collection

A QueryDBRecord collection is returned from the FindQueryDBRecords session method.

See Chapter 33, "Query Classes," FindQueryDBRecords, page 1910.

QueryDBRecord Collection Methods

In this section, we discuss the QueryDBRecord collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first QueryDBRecord object in the QueryDBRecord collection.

Parameters

None.

Returns

A reference to a QueryDBRecord object if successful, NULL otherwise.

Example

```
&MyQueryDBRecord = &MyCollection.First();
```

Item

Syntax

Item*(number)*

Description

The Item method returns the QueryDBRecord object that exists at the *number* position in the QueryDBRecord collection.

Parameters

Parameter	***Description***
Number	Specify the position number in the collection of the QueryDBRecord object that you want returned.

Returns

A reference to a QueryDBRecord object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryDBRecordColl.Count;
   &MyQueryDBRecord = &QueryDBRecordColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the QueryDBRecord object with the name *Name*.

Parameters

Parameter	***Description***
Name	Specify the name of an existing QueryDBRecord within the QueryDBRecord collection. If you specify an invalid name, the object is NULL.

Returns

A reference to a QueryDBRecord object if successful, NULL otherwise.

Example

```
&MyQueryDBRecord = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryDBRecord object in the QueryDBRecord collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryDBRecord object if successful, NULL otherwise.

Example

```
&MyQueryDBRecord = &MyCollection.Next();
```


QueryDBRecord Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryDBRecord objects in the QueryDBRecord Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryDBRecord Class

A QueryDBRecord object is returned from the QueryDBRecord Collection methods First, Item, ItemByName, or Next.

See Chapter 33, "Query Classes," QueryDBRecord Collection, page 2042.

QueryDBRecord Class Methods

In this section, we discuss the QueryDBRecord class methods. The methods are discussed in alphabetical order.

QueryDBRecordFieldByIndex

Syntax

QueryDBRecordFieldByIndex*(Index)*

Description

The QueryDBRecordFieldByIndex method returns the QueryDBRecordField object that exists at the *index* position in the QueryDBRecordField collection.

Parameters

Parameter	*Description*
Index	Specify the position number in the collection of the QueryDBRecordField object that you want returned.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

See Also

Chapter 33, "Query Classes," QueryDBRecordFieldByName, page 2046 and Chapter 33, "Query Classes," QueryDBRecordField Class, page 2052

QueryDBRecordFieldByName

Syntax

`QueryDBRecordFieldByName``(`*`Name`*`)`

Description

The QueryDBRecordFieldByName method returns the QueryDBRecordField object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of the QueryDBRecordField object that you want returned.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

See Also

Chapter 33, "Query Classes," QueryDBRecordFieldByIndex, page 2045 and Chapter 33, "Query Classes," QueryDBRecordField Class, page 2052

QueryDBRecord Class Properties

In this section, we discuss the QueryDBRecord class properties. The properties are discussed in alphabetical order.

Description

Description

This property returns the description of the QueryDBRecord as a string.

This property is read-only.

Name

Description

This property returns the name of the QueryDBRecord as a string.

This property is read-only.

QueryDBRecordFields

Description

This property returns a reference to a QueryDBRecordField Collection.

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryDBRecordField Collection, page 2048

RecordHierachy

Description

This property returns a reference to a QueryRecordHierarchy Collection.

The record hierarchy is not related to the query tree hierarchy shown when viewing access groups. Instead, it reflects an actual relationship between the record components, as defined in Application Designer using the Parent Record Name feature.

This property is read-only.

See Also

Chapter 33, "Query Classes," QueryRecordHierarchy Collection, page 2027

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query, "PeopleSoft Query Security"

QueryDBRecordField Collection

A QueryDBRecordField collection is returned from the QueryDBRecordFields QueryDBRecord property.

See Chapter 33, "Query Classes," QueryDBRecordFields, page 2047.

QueryDBRecordField Collection Methods

In this section, we discuss the QueryDBRecordField collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first QueryDBRecordField object in the QueryDBRecordField collection.

Parameters

None.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

Example

```
&MyQueryDBRecordField = &MyCollection.First();
```

Item

Syntax

Item*(number)*

Description

The Item method returns the QueryDBRecordField object that exists at the *number* position in the QueryDBRecordField collection.

Parameters

Parameter	*Description*
Number	Specify the position number in the collection of the QueryDBRecordField object that you want returned.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

Example

```
For &I = 1 to &Coll.Count;
   &MyQueryDBRecordField = &Coll.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the QueryDBRecordField object with the name *Name.*

Parameters

Parameter	*Description*
Name	Specify the name of an existing QueryDBRecordField within the QueryDBRecordField collection. If you specify an invalid name, the object is NULL.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

Example

```
&MyQueryDBRecordField = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryDBRecordField object in the QueryDBRecordField collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

Example

```
&MyQueryDBRecordField = &MyCollection.Next();
```


Sort

Syntax

Sort(*SortCriteria*)

Description

Use the Sort method to sort the fields within the QueryDBRecordField collection, based on the sort criteria specified with the method.

Parameters

Parameter	*Description*
SortCriteria	Specify the sort order for the list. This parameter can take either a constant or numeric value. See below.

The values for *OutputFormat* can be as follows:

Numeric Value	*Constant Value*	*Description*
1	%Query_SortNameAsc	Sort database fields in ascending order based on field name.
2	%Query_SortNameDesc	Sort database fields in descending order based on field name.
3	%Query_SoftFldNumAsc	Sort database fields in ascending order based on field number.
4	%Query_SortFldNumDesc	Sort database fields in descending order based on field number.

Returns

0

QueryDBRecordField Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryDBRecordField objects in the QueryDBRecordField Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryDBRecordField Class

A QueryDBRecordField object is from the QueryDBRecord Collection methods First, Item, ItemByName or Next.

See Chapter 33, "Query Classes," QueryDBRecord Collection, page 2042.

QueryDBRecordField Class Method

In this section, we discuss the QueryDBRecordField class methods in alphabetical order.

GetImageFormat

Syntax

GetImageFormat()

Description

Use the GetImageFormat method to differentiate between images and files, both stored as BLOBs in the database.

Note. Because images and files share the field type value 8, this method is required to differentiate between the two types.

Parameters

None.

Returns

A number. A value 16 indicates that the field is of type file (or attachment).

See Also

Chapter 33, "Query Classes," Type, page 2057

QueryDBRecordField Class Properties

In this section, we discuss the QueryDBRecordField class properties. The properties are discussed in alphabetical order.

Decimal

Description

This property returns the decimal positions for a field. This indicates how many numbers are allowed on the right side of the decimal.

This property is read-only.

Example

```
&FldDecs = &QryDBRcdFlds.Decimal;
```

Description

Description

This property returns the long description of the field as a string.

This is property is read-only.

Flag

Description

This property returns the Use and Edit Flags of the field, which are set when the Field is defined in Application Designer, as a numeric value.

Values for Use Flags are:

Value	***Description***
1	Key
2	Duplicate Key
4	System
8	Audit Field Add
16	Alternate Key
32	List Item
64	Descending Key
128	Audit Field Change
1024	Audit Field Delete
2048	Search Item
32768	Auto Update

Values for Edit Flags are:

Value	***Description***
256	Required
512	Edit Translate
4096	Date Range Edit
8192	Yes/No Table Edit
16384	Edit Table
262144	From Search Field

Value	*Description*
524288	Through Search Field
8388608	Use Default Label Flag
16777216	Default Search Field

This is a read-only property.

Format

Description

This property returns the field format for a field. Values are:

Value	*Description*
1	No format
2	Name
3	Phone Number North America
4	Zip/Postal Code North America
5	Social Security Number (SSN)
6	Uppercase
7	Mixed case
8	Raw binary
9	Numbers only
10	Canadian Social Insurance Number (SIN)
11	Phone Number International
12	Zip/Postal Code International
13	Seconds
14	Microseconds
15	Custom

This property is read-only.

Length

Description

This property returns the length of the field as a number.

This property is read-only.

LongName

Description

This property returns the long name of the field as a string.

This property is read-only.

LookupTableName

Description

If the field has a lookup table associated with it, this property returns the name of that Lookup Table, else it returns an empty string.

This is a read-only property.

LookupTableRecord

Description

If the field has a lookup table associated with it, this property returns the instance of the QueryDBRecord for that Lookup Table, else it returns NULL.

This is a read-only property.

Name

Description

This property returns the name of the field as a string.

This property is read-only.

Shortname

Description

This property returns the short name of the field as a string.

This property is read-only.

Type

Description

This property returns the type of the field. You can specify either a constant or a numeric value for this property. The values are:

Numeric Value	***Constant Value***	***Description***
0	%FieldType_Char	Character
1	%FieldType_LongChar	Long Character
2	%FieldType_Number	Number
3	%FieldType_SignedNumber	Signed number
4	%FieldType_Date	Date
5	%FieldType_Time	Time
6	%FieldType_DateTime	Datetime
8	%FieldType_Image	Image
8	%FieldType_File	File
9	%FieldType_ImageRef	ImageReference

Note. The Image and File types can be differentiated using the GetImageFormat method of the QueryField class.

This property is read-only.

See Also

Chapter 33, "Query Classes," GetImageFormat, page 2052

QueryPrompt Collection

A QueryPrompt collection is returned from the Prompts and RuntimePrompts Query class properties.

See Chapter 33, "Query Classes," Prompts, page 1953 and Chapter 33, "Query Classes," RunTimePrompts, page 1954.

QueryPrompt Collection Methods

In this section, we discuss the QueryPrompt collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first QueryPrompt object in the QueryPrompt collection.

Parameters

None.

Returns

A reference to a QueryPrompt object if successful, NULL otherwise.

Example

```
&MyQueryPrompt = &MyCollection.First();
```

Item

Syntax

Item*(number)*

Description

The Item method returns the QueryPrompt object that exists at the *number* position in the QueryPrompt collection.

Parameters

Parameter	*Description*
Number	Specify the position number in the collection of the QueryPrompt object that you want returned.

Returns

A reference to a QueryPrompt object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryPromptColl.Count;
   &MyQueryPrompt = &QueryPromptColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the QueryPrompt object with the name *Name*.

Parameters

Parameter	*Description*
Name	Specify the name of an existing QueryPrompt within the QueryPrompt collection. If you specify an invalid name, the object is NULL.

Returns

A reference to a QueryPrompt object if successful, NULL otherwise.

Example

```
&MyQueryPrompt = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryPrompt object in the QueryPrompt collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryPrompt object if successful, NULL otherwise.

Example

```
&MyQueryPrompt = &MyCollection.Next();
```


QueryPrompt Collection Property

In this section, we discuss the QueryPrompt collection properties. The properties are discussed in alphabetical order.

Count

Description

This property returns the number of QueryPrompt objects in the QueryPrompt Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryPrompt Class

A reference to a QueryPrompt is returned by the following:

- The First, Item, ItemByName, and Next methods of a QueryPrompt Collection.
- The AddPrompt Query class method.

See Chapter 33, "Query Classes," QueryPrompt Collection, page 2058 and Chapter 33, "Query Classes," AddPrompt, page 1922.

See *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Query*, "Defining Selection Criteria," Defining Prompts.

QueryPrompt Properties

In this section, we discuss the QueryPrompt properties. The properties are discussed in alphabetical order.

EditType

Description

This property returns or sets the edit type for the field. This property takes a number value. The values are:

Value	*Description*
0	No table edit
16384	Prompt table
512	Translate table
8192	Yes/No

This property is read-write.

FieldDecimal

Description

This property returns or sets the decimal value for the field.

This property is only valid with number fields.

This property is read-write.

FieldFormat

Description

This property returns the field format for a field. This property takes a number value. Values are:

Value	*Description*
0	None
1	Name
2	Phone
3	Zip Code
4	Social Security Number
5	Upper
6	Mixed Case
7	Century

Value	*Description*
8	Number Only
9	Social Insurance Number
10	International Phone Number
11	International Postal Code
12	Seconds
13	Microseconds
14	Century/Seconds
15	Century/Microseconds

This property is read-only.

FieldLength

Description

This property returns or sets the field length.

This property is read-write.

FieldName

Description

This property returns or sets the field name used with the prompt.

This property is read-write.

FieldType

Description

This property returns or sets the field type of the field used with the prompt.

This property returns the type of the field. You can specify either a constant or a numeric value. The values are:

Numeric Value	Constant Value	Description
0	%FieldType_Char	Character
1	%FieldType_LongChar	Long Character
2	%FieldType_Number	Number
3	%FieldType_SignedNumber	Signed number
4	%FieldType_Date	Date
5	%FieldType_Time	Time
6	%FieldType_DateTime	Datetime
7	%FieldType_Image	Image

This property is read-write.

HeadingText

Description

This property returns or sets the heading text for the prompt field.

This property is read-write.

HeadingType

Description

This property returns or sets the heading type for the query field. This property takes either a constant or numeric value. The values are:

Numeric Value	Constant Value	Description
1	%Query_HdgNone	Query field has no heading.
2	%Query_HdgText	Query field has a text heading.
3	%Query_HdgRftShort	Query field uses the short RFT heading.
4	%Query_HdgRftLong	Query fields uses the long RFT heading.

This property is read-write.

LangCount

Description

This property returns the total count of the language records for the current prompt.

This property is read-only.

Name

Description

This property returns a string containing the Prompt name. When an existing query is read, this name is the same as the Field Name. When a new prompt is added using AddPrompt, this is the *Name* parameter used with that method.

This property is read-only.

PromptRecordFieldName

Description

When you have more than one field with the same name in the prompt collection, the system generates a unique prompt field name for each repeated field. The generated names are of the form Bind1, Bind2, and so on. This property returns the unique name for the prompt record field.

This property is read-only.

PromptTable

Description

This property returns or sets the prompt table name for the prompt field.

This property is read-write.

UniquePromptName

Description

This property returns or sets the unique prompt name for the prompt field.

This property is read-write.

UseCount

Description

This property returns the total count of the usage of the current prompt.

This property is read-only.

The following are examples of the usual actions that you perform using the query classes.

Query Classes Examples

The following are examples of the usual actions that you perform using the query classes.

Creating a New Query

In this example, you are creating a new query, adding a record and two fields. The following is the complete code sample: the steps explain each line.


```
Local ApiObject &aQuery, &aQrySelCol;
Local ApiObject &COLL, &ERROR;
Local String &TEXT;
Local Session &MySession;
Local Record &aQryRcd;
Local Field &aQryFld;

&MySession = %Session;

If &MySession <> Null Then

   &aQuery = &MySession.GetQuery();

   &aQuery.Create("TEST1", False, %Query_Query, "PIA Test 1", ⇒
"Creating Test Query 1 from PIA Page");

   &aQrySel = &aQuery.AddQuerySelect();

   &aQryRcd = &aQrySel.AddQueryRecord("ABSENCE_HIST");
   &aQryFld = &aQrySel.AddQuerySelectedField("ABSENCE_HIST", "A", "EMPLID", "ID");

   If &aQryFld <> Null Then
         &aQryFld.ColumnNumber = 1;
         &aQryFld.HeadingType = %Query_HdgRftShort;
      End-If;

   &Rslt = &aQuery.Save();

   If &Rslt <> 0 Then
      /* save didn't complete */
      &COLL = &MySession.PSMessages;

      For &I = 1 to &COLL.Count
         &ERROR = &COLL.Item(&I);
         &TEXT = &ERROR.Text;
         /* do error processing */
      End-For;

      &COLL.DeleteAll();
   End-if;
    /* error processing for not getting a session */
End-if;
```

To create a new query:

1. Get a session object.

   Before you can create a query, you have to get a session object. The session controls access to the query, provides error tracing, enables you to set the runtime environment, and so on. Then this program checks to verify that the session object is valid.

   ```
   &MySession = %Session;

   &aQuery = &MySession.GetQuery();

   If &MySession <> Null Then
   ```

2. Create the query.

   Use the Create method to create the query. This query is a private query, of type query.

   ```
   &aQuery.Create("TEST1", False, %Query_Query, "Test 1", "Creating Test Query");
   ```

3. Add a QuerySelect.

   The QuerySelect contains the main query statement for the query. There can be multiple QuerySelect objects for queries that involve unions or subqueries. Each select (or union or subquery) consists of QueryRecords, QueryOutputFields, QuerySelectedFields, and QueryCriteria and is treated as a child of the MAIN select statement.

```
&aQrySel = &aQuery.AddQuerySelect();
```

4. Add a record and a field.

   The AddQueryRecord method adds a query record to the query. The AddQuerySelectedField adds a field, using the record alias "A". The ID is what gets displayed in the heading for the query.

```
&aQryRcd = &aQrySel.AddQueryRecord("ABSENCE_HIST");
&aQryFld = &aQrySel.AddQuerySelectedField("ABSENCE_HIST", "A", "EMPLID", "ID");
```

5. Make the field an output field.

   The field was added as a selected field. By setting the ColumnNumber to a number greater than one, the field is now an output field. The text that's displayed in the heading comes from the RFT short description of the field.

```
      If &aQryFld <> Null Then
         &aQryFld.ColumnNumber = 1;
         &aQryFld.HeadingType = %Query_HdgRftShort;
      End-If;
```

6. Save the data.

   When you execute the Save method, the new query is saved to the database.

```
&Rslt = &aQuery.Save();
```

```
If &Rslt <> 0 Then
```

   The Save method returns a numeric value: 0 if successful. You can use this value to do error checking.

7. Check Errors.

   You can check if there were any errors using the PSMessages property on the session object.

```
   /* save didn't complete */
   &COLL = &MySession.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, you may want to delete it from the PSMessages collection.

Adding Criteria

In this example, you are accessing an existing query, then adding criteria both as part of the query as well as part of a subquery. The SQL statement created by this subquery is as follows:


```
SELECT RECNAME, RECDESCR, RELLANGRECNAME, PARENTRECNAME, DESCRLONG from PSRECDEFN ⇒
where RECNAME IN (select OBJECTVALUE1 from PSPROJECTITEM where PROJECTNAME =⇒
 'PPLTOOLS') AND RECTYPE = 0 order by RECNAME
```

The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MyQuery, &MainQrySel, &Criteria1, &MyCrit2Expr2, &MyCriteria2;
Local ApiObject &SubQrySel, &SubQryCrit1, &SubQryExpr1, &MyCrit2Expr2;
Local Record &SubQryRec;
Local Session = &MySession;
Local ApiObject &COLL, &ERROR;
Local String &TEXT;

&MySession = %Session;

If &MySession <> Null Then

&MyQuery = &MySession.GetQuery();

&MyQuery.Open("Table", False, True);

&MainQrySel = &MyQuery.QuerySelect;

/* Add query record, add fields, then make selected fields output fields */

&MainRec = &MainQrySel.AddQueryRecord("PSRECDEFN");

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "RECNAME", "Record Name");
&QryFld.ColumnNumber = 1;
&QryFld.OrderByNumber = 1;

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "RECDESCR", "Record Descr");
&QryFld.ColumnNumber = 2;
&QryFld.OrderByNumber = 2;

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "RELLANGRECNAME", "Record Lang Rec");
&QryFld.ColumnNumber = 3;
&QryFld.OrderByNumber = 3;

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "PARENTRECNAME", "Parent Record Name");
&QryFld.ColumnNumber = 4;

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "DESCRLONG", "Long Descr");
&QryFld.ColumnNumber = 5;

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "EMPLID", "ID");
&QryFld.ColumnNumber = 6;

/* adding first criteria */
&Criteria1 = &MainQrySel.AddCriteria("FirstCriteria");

/* First criteria will not have any logical AND/OR */
&Criteria1.Logical = %Query_CombNotUsed;

&Criteria1.Expr1Type = %Query_ExprField;
&Criteria1.AddExpr1Field(&MainRec.RecordAlias, "RECNAME");

/* So that the criteria is constructed as - RECNAME IN (...)*/
&Criteria1.Operator = %Query_CondInList;
&Criteria1.Expr2Type = %Query_ExprSubQuery;

&SubQrySel = &Criteria1.AddExpr2SubQuery();
```



```
&SubQryRec = AddQueryRecord("PSPROJECTITEM");
&SubQryFld1 = &SubQrySel.AddQuerySelectedField(&SubQryRec.Name, &SubQryRec.Record⇒
Alias, "OBJECTVALUE1", "Join Object")
&SubQryFld1.ColumnNumber = 1;

/* Need criteria - PROJECTNAME = 'PPLTOOLS'  - in the subquery */
&SubQryCrit1 = &SubQrySel.AddCriteria("FirstSubCrit");

/* First criteria will not have any logical AND/OR */
&SubQryCrit1.Logical = %Query_CombNotUsed;

&SubQryCrit1.Expr1Type = %Query_ExprField;
&SubQryCrit1.AddExpr1Field(&SubQryRec.RecordAlias, "PROJECTNAME");
&SubQryCrit1.Operator = %Query_CondEqual;

/* So that the criteria is constructed as - PROJECTNAME = 'PPLTOOLS'*/
&SubQryCrit1.Expr2Type = %Query_ExprConstant;
&SubQryExpr1 = &SubQryCrit1.AddExpr2Expression();
&SubQryExpr1.Text = "PPLTOOLS";
&SubQryCrit1.Expr2Expression1 = &SubQryExpr1;

/* Second Criteria, which is for RECTYPE = 0 */
&MyCriteria2 = &MainQrySel.AddCriteria("SecondCriteria");

&MyCriteria2.Expr1Type = %Query_ExprField;
&MyCriteria2.AddExpr1Field(&MainRec.RecordAlias, "RECTYPE");

/* Since this is second criteria, we need a logical AND to state that */
/* - AND RECTYPE = 0 */

&MyCriteria2.Logical = %Query_CombAnd;

&MyCriteria2.Operator = %Query_CondEqual;
&MyCriteria2.Expr2Type = %Query_ExprConstant;
&MyCrit2Expr2 = &MyCriteria2.AddExpr2Expression();
&MyCriteria2.Expr2Expression1 = &MyCrit2Expr2;
&MyCrit2Expr2.Text = "0";

&Rslt = &MyQuery.Save();

If &Rslt <> 0 Then
   /* save didn't complete */
   &COLL = &MySession.PSMessages;

      For &I = 1 to &COLL.Count
         &ERROR = &COLL.Item(&I);
         &TEXT = &ERROR.Text;
         /* do error processing */
      End-For;

      &COLL.DeleteAll();
   End-if;

Else

/* do error processing for not getting session */

End-if;
```

To add criteria to a query:

1. Get a session object.

   Before you can create a query, you have to get a session object. The session controls access to the query, provides error tracing, enables you to set the runtime environment, and so on. Then this program checks to verify that the session object is valid.

   ```
   &MySession = %Session;
   ```

   ```
   &MyQuery = &MySession.GetQuery();
   ```

   ```
   If &MySession <> Null Then
   ```

2. Access an existing query and get the main query select statement.

   Use the Open method to get the existing query. Then access the main select statement with the QuerySelect property.

   ```
   &MyQuery.Open("Table", False, True);
   ```

   ```
   &MainQrySel = &MyQuery.QuerySelect;
   ```

3. Add Query Record in the Main Select.

   Add the query record that you want to use.

   ```
   &MainRec = &MainQrySel.AddQueryRecord("PSRECDEFN");
   ```

4. Add the displayed fields.

   You want to add the selected fields. Note instead of hardcoding the name of the record, this code example uses the Name property. Also, the code uses the RecordAlias property instead of hardcoding the alias. This makes the code easier to read, as well as easier to maintain. Specifying a column number also makes this an output field.

   ```
   &QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
    "RECNAME", "Record Name");
   &QryFld.ColumnNumber = 1;
   ```

5. Specify the OrderBy value.

   Because we need to order by this field, the OrderByNumber of that field must be set also.

   ```
   &QryFld.OrderByNumber = 1;
   ```

6. Add the first criteria.

   Add the first criteria. You don't want it added with any kind of operator, like an AND or an OR, so the Logical property of the first criteria is set with %Query_CombNotUsed. This is also used because it's the first non-having criteria of a query.

   ```
   /* adding first criteria */
   &Criteria1 = &MainQrySel.AddCriteria("FirstCriteria");
   ```

   ```
   /* First criteria will not have any logical AND/OR */
      &Criteria1.Logical = %Query_CombNotUsed;
   ```


7. Add the first criteria field.

   The first field for the criteria is a QueryExpression type field. The type of the field *must* be set before the field is added.

   ```
   &Criteria1.Expr1Type = %Query_ExprField;
   &Criteria1.AddExpr1Field(&MainRec.RecordAlias, "RECNAME");
   ```

8. Add the condition for the first criteria and the subquery.

   The first criteria is the WHERE RECNAME IN portion of the SQL statement. The condition is considered 'in list', where the list is the result of the subquery. The expression is a subquery. Again, you have to set the type again before adding the subquery.

   ```
   &Criteria1.Operator = %Query_CondInList;
   &Criteria1.Expr2Type = %Query_ExprSubQuery;
   &SubQrySel = &Criteria1.AddExpr2SubQuery();
   ```

9. Add the records for the subquery.

   Add the query record and the field from the query field, and make it an output field.

   ```
   &SubQryRec = AddQueryRecord("PSPROJECTITEM");
   &SubQryFld1 = &SubQrySel.AddQuerySelectedField(&SubQryRecName.Name, ⇒
   &SubQryRec.RecordAlias, "OBJECTVALUE1", "Join Object");
   &SubQryFld1.ColumnNumber = 1;
   ```

10. Add the criteria in the subquery.

    The following code adds the criteria for the subquery. Because this is the first non-having criteria in a select statement, the Logical property is set as %Query_CombNotUsed. Then the first expression is added as a field, and set to be equal to the second expression, PPLTOOLS. This is the where PROJECTNAME = 'PPLTOOLS' portion of the SQL statement.

    ```
    /* Need criteria - PROJECTNAME = 'PPLTOOLS'  - in the subquery */
    &SubQryCrit1 = &SubQrySel.AddCriteria("FirstSubQryCrit");

    /* First criteria will not have any logical AND/OR */
       &SubQryCrit1.Logical = %Query_CombNotUsed;

       &SubQryCrit1.Expr1Type = %Query_ExprField;
       &SubQryCrit1.AddExpr1Field(&SubQryRec.RecordAlias, "PROJECTNAME");
    &SubQryCrit1.Operator = %Query_CondEqual;

    /* So that the criteria is constructed as - PROJECTNAME = 'PPLTOOLS'*/
    &SubQryCrit1.Expr2Type = %Query_ExprConstant;
    &SubQryExpr1 = &SubQryCrit1.AddExpr2Expression();
    &SubQryExpr1.Text = "PPLTOOLS";
       &SubQryCrit1.Expr2Expression1 = &SubQryExpr1;
    ```

11. Add the second criteria to the main select.

    Add the second criteria. Remember to set the type for the expression field first. Because this is the second criteria, we need a logical AND to state that this criteria is used with the first criteria.

    ```
    /* Second Criteria, which is for RECTYPE = 0 */
    &MyCriteria2 = &MainQrySel.AddCriteria("SecondCriteria");

    &MyCriteria2.Expr1Type = %Query_ExprField;
    &MyCriteria2.AddExpr1Field(&MainRec.RecordAlias, "RECTYPE");

    /* Since this is second criteria, we need a logical AND to state that*/
    /* - AND RECTYPE = 0 */

    &MyCriteria2.Logical = %Query_CombAnd;

    &MyCriteria2.Operator = %Query_CondEqual;
    &MyCriteria2.Expr2Type = %Query_ExprConstant;

    &MyCrit2Expr2 = &MyCriteria2.AddExpr2Expression();
    &MyCriteria2.Expr2Expression1 = &MyCrit2Expr2;
    &MyCrit2Expr2.Text = "0";
    ```

12. Save the data.

    When you execute the Save method, the new query is saved to the database.

    ```
    &Rslt = &MyQuery.Save();
    ```

    ```
    If &Rslt <> 0 Then
    ```

    The Save method returns a numeric value: 0 if successful. You can use this value to do error checking.

13. Check Errors

    You can check if there were any errors using the PSMessages property on the session object.

    ```
       /* save didn't complete */
       &COLL = &MySession.PSMessages;
       For &I = 1 to &COLL.Count
          &ERROR = &COLL.Item(&I);
          &TEXT = &ERROR.Text;
          /* do error processing */
       End-For;
       &COLL.DeleteAll();
    End-if;
    ```

    If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, you may want to delete it from the PSMessages collection.

Using Outer Joins

The following PeopleCode query uses outer joins:


```
   Local ApiObject &aQuery, &aQrySelCol;
   Local ApiObject &aQryRcd, &aQryRcd2;
   Local ApiObject &aQryFld, &aQryFld2;
   Local ApiObject &aQrySel, &Criteria1;
   Local number &Rslt;

   &aQuery = %Session.GetQuery();
   &aQuery.Create("TEST1", False, %Query_Query, "PIA Test 1", ⇒
"Creating Test Query1 from PIA Page");

   &aQrySel = &aQuery.AddQuerySelect();

   &aQryRcd = &aQrySel.AddQueryRecord("JOB");
   &aQryRcd.RecordAlias = "A";

   &aQryRcd2 = &aQrySel.AddQueryRecord("PERSONAL_DATA");
   &aQryRcd2.RecordAlias = "B";
   &aQryRcd2.JoinType = %Query_JoinLeftOuter;
   &aQryRcd2.JoinAlias = "A";

   &aQryFld = &aQrySel.AddQuerySelectedField("JOB", "A", "EMPLID", "EMPLID");
   &aQryFld.ColumnNumber = 1;
   &aQryFld.HeadingType = %Query_HdgRftShort;

   &aQryFld2 = &aQrySel.AddQuerySelectedField("PERSONAL_DATA", "B", ⇒
"NAME", "NAME");
   &aQryFld2.ColumnNumber = 2;
   &aQryFld.HeadingType = %Query_HdgRftShort;

   &Criteria1 = &aQrySel.AddCriteria("JoinCriteria");
   &Criteria1.Logical = %Query_CombNotUsed;
   &Criteria1.Expr1Type = %Query_ExprField;
   &Criteria1.AddExpr1Field("A", "EMPLID");
   &Criteria1.Operator = %Query_CondEqual;
   &Criteria1.Expr2Type = %Query_ExprField;
   &Criteria1.AddExpr2Field1("B", "EMPLID");
   &Criteria1.OJAlias = "B";

   &Rslt = &aQuery.Save();
```

The above PeopleCode program produces the following SQL:

```
SELECT A.EMPLID, B.NAME
FROM (PS_JOB A LEFT OUTER JOIN PS_PERSONAL_DATA B ON A.EMPLID=B.EMPLID)
```


Chapter 34

Record Class

This chapter provides an overview of Record class and discusses the following topics:

- Shortcut considerations.
- Record methods used to create SQL statements.
- Record object declaration.
- Scope of a record object.
- Record class built-in functions.
- Record class reference.

Understanding Record Class

A record object, instantiated from the Record class, is a single instance of a data within a row and is based on a record definition. A record object consists of one to *n* fields.

If a record object is instantiated using GetRecord (either the function or the method), the record object that is instantiated references the record from the current row in the data buffers, and its associated data.

If a record object is instantiated using the CreateRecord function, the record object that's instantiated is a freestanding record definition with its component set of field objects in the data buffer. The fields created by this function are initialized to null values, that is, they do *not* contain any data. Default processing is not performed. You can select into this record object using the SelectByKey method. You can also select into this record object using SQLExec.

CopyFieldsTo is a commonly used method for the record class. Name, IsChanged, and FieldCount are commonly used properties.

The record class is one of the data buffer access classes.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Shortcut Considerations

An expression of the form

RECORD.*recname.property*

or

RECORD.*recname.method*(. . .)

is converted to an object expression by using **GetRecord**(**RECORD.***recname*). For example, the following two lines of code are equivalent:

```
&MyRow = RECORD.EMPL_CHKLST_ITEM.ParentRow;
```

```
&MyRow = GetRecord(RECORD.EMPL_CHKLST_ITEM).ParentRow;
```

In addition, the default method for the Record class is the GetField method. This means you can access a field by just specifying the field name, instead of using GetField. For example, the following two lines of code, which both disable the EMPLID field, are equivalent:

```
&MyRec.EMPLID.Enabled = False;
```

```
&MyRec.GetField(FIELD.EMPLID).Enabled = False;
```

Note. If the field you're accessing has the same name as a record property (such as, Name) you can't use this method for accessing the field. You must use the GetField method.

Record Methods Used to Create SQL Statements

You can use the following record object methods to build and execute SQL statements:

- Delete
- SelectByKey
- Insert
- Save
- Update

The methods that result in a database update (specifically, UPDATE, INSERT, and DELETE) can only be issued in the following events:

- SavePreChange
- WorkFlow
- SavePostChange
- FieldChange

Remember that record UPDATEs, INSERTs, and DELETEs go directly to the database server, not to the Component Processor (although you can *view* data in the buffer using the PeopleCode debugger and other data buffer access classes). If a record method assumes that the database has been updated based on changes made in the component, that record method can be issued only in the SavePostChange event, because before SavePostChange none of the changes made to page data has actually been written back to the database.

If your application is repeating the same instruction many times, such as doing a million INSERTs, you should use the SQL object with the BulkMode property set to True, rather than the record SQL methods.

See Also

Chapter 34, "Record Class," Delete, page 2085

Chapter 34, "Record Class," SelectByKey, page 2096

Chapter 34, "Record Class," Insert, page 2090

Chapter 34, "Record Class," Update, page 2102

Chapter 40, "SQL Class," page 2239

Data Type of a Record Object

Record objects are declared as type Record. For example,

```
Local Record &MYRECORD;
```

Scope of a Record Object

A record can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, Component Interface PeopleCode, and so on.

Record Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRecord

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRecord

Record Class Methods

In this section, we discuss each Record class method. The methods are discussed in alphabetical order.

CompareFields

Syntax

`CompareFields`*`(recordobject)`*

Description

The CompareFields method compares all like-named fields of the record object executing the method with the specified record object *recordobject.*

Parameters

Parameter	***Description***
recordobject	Specify a record object for use in the comparison. The specified record object does not have to refer to the same record as the record object executing the method.

Returns

A Boolean value; True if all like-named fields have the same value.

Example

```
&REC = GetRecord(RECORD.OP_METH_VW);
&REC2 = GetRecord(RECORD.OPC_METH);
If &REC2.CompareFields(&REC) Then
    WinMessage("All liked named fields have the same value");
End-If;
```

See Also

Chapter 34, "Record Class," CopyChangedFieldsTo, page 2081 and Chapter 34, "Record Class," CopyFieldsTo, page 2082

CopyChangedFieldsTo

Syntax

`CopyChangedFieldsTo`*`(recordobject)`*

Description

The CopyChangedFieldsTo method copies all like-named field values that have changed from the record object executing the method to the specified record object *recordobject.* This copies only changed field values. To copy all field values, use the CopyFieldsTo method.

Note. This method works only with database records. The Component Processor doesn't track the contents of work records, so there is no changed value to use for copying changed fields.

Parameters

Parameter	***Description***
recordobject	Specify a record object to be copied to. The specified record object does not have to refer to the same record as the record object executing the method.

Returns

None.

Example

```
Local Record &REC, &REC2;

&REC = GetRecord(RECORD.OPC_METH);

/* make changes to the values of fields in the record */

&REC2 = CreateRecord(RECORD.OPC_METH_WORK);

&REC.CopyChangedFieldsTo(&REC2);
```

See Also

Chapter 34, "Record Class," CompareFields, page 2080 and Chapter 34, "Record Class," CopyFieldsTo, page 2082

CopyFieldsTo

Syntax

```
CopyFieldsTo(recordobject[, DontCopyUnusedInSource [,
DontCopyUnusedInDestination]] [,IsBatch])
```

Description

The CopyFieldsTo method copies all like-named field values from the record object executing the method to the specified record object *recordobject.* This copies all field values.

To copy only changed field values, use the CopyChangedFieldsTo method.

To restrict the copy to fields that have been marked as unused (set using the SetDBFieldNotUsed function) you can specify either *DontCopyUnusedInSource* or *DontCopyUnusedInDestination.*

Note. If you are copying to a derived work record, the IsChanged flag for the record is not set. Copying fields to a database record does set the IsChanged flag to True.

Parameters

Parameter	*Description*
recordobject	Specify a record object to be copied to. The specified record object does not have to refer to the same record as the record object executing the method.
DontCopyUnusedInSource	Specify this parameter to be True if you want to restrict the copy of fields to not include fields marked as unused in the record of the object performing the CopyFieldsTo method. This is the source record. The default value for this parameter is False, which means that there are no restrictions on the copy of unused fields as defined in the source record.
DontCopyUnusedInDestination	Specify this parameter to be True if you want to restrict the copy of fields to not include fields marked as unused in the recordobject record which is the destination record. **Note.** To specify this record to be true you must specify a value for *DontCopyUnusedInSource.* The default value for this parameter is False meaning that there are no restrictions on the copy of unused fields as defined in the destination record.
IsBatch	Specify whether this method is being called from an online program or from a batch program (such as an Application Engine program.) The default is false. Using this method may improve batch performance. When set to true, the system uses the FieldNotUsed information directly from the field properties already in the memory

Returns

None.

Example

```
Local Record &REC, &REC2;

&REC = GetRecord(RECORD.OPC_METH);
&REC2 = CreateRecord(RECORD.OPC_METH_WORK);
&REC.CopyFieldsTo(&REC2);
```

In the following example, records are copied into a record after being fetched.

```
Component number &displayNum;
Local SQL &sql;
Local Rowset &rs, &rs2;

&level0 = GetLevel0();
&displayNum = WS_NUM_ORDERS;

&rs = GetRowset(Record.WS_ORD_HDR_VW);
&rs.Flush();
WinMessage("1");
&sql = CreateSQL("%selectall(:1) where BUSINESS_UNIT=:2", Record.WS_ORD_HDR_VW,⇒
 "M04");
WinMessage("2");
&rec = CreateRecord(Record.WS_ORD_HDR_VW);

For &i = 1 To &displayNum

   If &sql.Fetch(&rec) Then

      &rs.InsertRow(&i);

      &rec.copyfieldsto(&rs.GetRow(&i).WS_ORD_HDR_VW);

      &rs2 = &rs.GetRow(&i).GetRowset(1);
      &rs2.Select(Record.WS_ORD_LINE_VW, "where BUSINESS_UNIT=:1 and ORDER_NO=:2",⇒
 &rec.BUSINESS_UNIT.value, &rec.ORDER_NO.value);

      /* Hide rows that do not contain Products */
      If &rs2 = Null Or None(&rs2.GetRow(1).WS_ORD_LINE_VW.ORDER_INT_LINE_⇒
NO.Value) Then
         For &j = 1 To &rs2.ActiveRowCount
            &rs2.GetRow(&j).Visible = False;
         End-For;
      End-If;
   End-If;
End-For;

&rs.GetRow(&i).Visible = False;
```

The following example is for unused record fields:


```
Local Record &From = CreateRecord(Record.QE_UPS_TIME);

/* setup the initial values */

&From.QE_FROM_ZIP.Value = "12345";
&From.QE_TO_ZIP.Value = "67890";
&From.QE_UPS_TIME_BUTTON.Value = "A";
&From.DESCRLONG.Value = "This is the from record.";

/* Now make one of the fields unused */
Local string &ToZip_Value = "77777";

/* start out clean with &To */
Local Record &To = CreateRecord(Record.QE_UPS_TIME);
&To.QE_TO_ZIP.Value = &ToZip_Value;
If SetDBFieldNotUsed(Field.QE_TO_ZIP, True) <> %MetaDataChange_Success Then
   MessageBox(0, "", 0, 0, "SSetDBFieldNotUsed(Field.QE_TO_ZIP, TRUE) fails??");
End-If;
/* Copy no unused fields from the source record */
&From.CopyFieldsTo(&To, True /* no unused fields from source */);

/* Copy to no unused fields in the destination */
&From.CopyFieldsTo(&To, False /* all fields from source */, True /* no unused⇒
 fields in dest */);

/* Copy no unused fields either in source or destination */

&From.CopyFieldsTo(&To, True /* no unused fields from source */, True /* no unused⇒
 fields in dest */);

/* Now finally make that field good again */

If SetDBFieldNotUsed(Field.QE_TO_ZIP, False) <> %MetaDataChange_Success Then
   MessageBox(0, "", 0, 0, "SSetDBFieldNotUsed(Field.QE_TO_ZIP, False) fails??");
End-If;
```

See Also

Chapter 34, "Record Class," CompareFields, page 2080; Chapter 34, "Record Class," CopyChangedFieldsTo, page 2081; Chapter 34, "Record Class," IsChanged, page 2105 and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," SetDBFieldNotUsed

DBPatternMatch

Syntax

DBPatternMatch(*Value*,*Pattern*,*CaseSensitive*)

Description

Use the DBPatternMatch function to match the string in *Value* to the given pattern.

You can use wildcard characters % and _ when searching. % means find all characters, while _ means find a single character. For example, to find if the string in *Value* started with the letter M, use "M%" for *Pattern.* To find either DATE or DATA, use "DAT_" for *Pattern.*

These characters can be escaped (that is, ignored) using a \. For example, to search for a value that contains the character %, use \% in *Pattern*.

If *Pattern* is an empty string, this function retrieves the value based only on the specified case sensitivity (that is, specifying "" for *Pattern* is the same as specifying "%").

Parameters

Parameter	***Description***
Value	Specify the string to be searched.
Pattern	Specify the pattern to be used when searching.
CaseSensitive	Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it is not.

Returns

A Boolean value: True if the string matches the pattern, False otherwise.

Delete

Syntax

```
Delete()
```

Description

The Delete method uses the key fields of the record and their values to build and execute a DELETE SQL statement which deletes the record (row of data) from the SQL data table.

This method, like the DeleteRow Rowset class method, initially marks the record or row as needing to be deleted. At save time the row is actually deleted from the database and cleared from the buffer.

Because this method results in a database change, it can be issued only in the following events:

- SavePreChange
- WorkFlow
- SavePostChange

If your application is repeating the same instruction multiple times, such as doing a million DELETEs, use the SQL object with the BulkMode property set to True, rather than the record SQL methods.

For every record deleted by the Delete method, if the set language is not the base language and the record has related language records, the Delete method tries to do related language processing.

Parameters

None.

Returns

The result is True on successful completion, False if the record was not found. Any other conditions cause termination.

Example

Suppose that KEYF1 and KEYF2 are the two key fields of record definition MYRECORD. The following code deletes the database record that has KEYF1 equal to "A" and KEYF2 equal to "X":

```
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
&REC.Insert();
```

See Also

Chapter 34, "Record Class," SelectByKey, page 2096; Chapter 34, "Record Class," Insert, page 2090 and Chapter 34, "Record Class," Update, page 2102

Chapter 36, "Rowset Class," DeleteRow, page 2133

Chapter 40, "SQL Class," page 2239

PeopleTools 8.51 PeopleBook: Global Technology, "Using Related Language Tables"

ExecuteEdits

Syntax

`ExecuteEdits``([`*`editlevel`*`]);`

where *editlevels* is a list of values in the form:

`editlevel1`` [+ `*`editlevel2`*`] . . .]);`

and where *editleveln* is one of the following constants:

%Edit_DateRange

%Edit_OneZero

%Edit_PromptTable

%Edit_Required

%Edit_TranslateTable

%Edit_YesNo

Description

The ExecuteEdits method executes the standard system edits on every field in the record. The types of edits performed depends on the *editlevel*. If no *editlevel* is specified, all system edits are executed. All *editlevels* are already defined for the record definition or for the field definition, that is:

- Reasonable Date Range (Is the date contained within the specified reasonable date range?)
- 1/0 (Do all 1/0 fields contain only a 1 or 0?)
- Prompt Table (Is field data contained in the specified prompt table?)
- Required Field (Do all required fields contain data? For numeric or signed fields, it checks that they do not contain NULL or 0 values.)
- Translate Table (Is field data contained in the specified translate table?)
- Yes/No (Do all yes/no fields only contain only yes or no data?)

Note. ExecuteEdits does not perform any validation on DateTime fields.

If any of the edits fail, the status of the property IsEditError is set to False for the record. The field property EditError is set to True for any fields that are in error. In addition, the Field class properties MessageNumber and MessageSetNumber are set to the number of the returned message and message set number of the returned message, for each field in error.

You must use the SetEditTable method to set the prompt tables for fields that are defined with %EditTable in the record definition.

If you're running an Application Engine program, and you want to do set based ExecuteEdits (as opposed to row-by-row) consider using the meta-SQL construct %ExecuteEdits.

See Chapter 34, "Record Class," SetEditTable, page 2100; Chapter 18, "Field Class," EditError, page 951; Chapter 18, "Field Class," MessageNumber, page 962; Chapter 18, "Field Class," MessageSetNumber, page 963 and *Enterprise PeopleTools 8.51 PeopleBook: Application Engine*, "Using Meta-SQL and PeopleCode," %ExecuteEdits.

Considerations for ExecuteEdits and SetEditTable

If an effective date is a key on the prompt table, and the record being edited doesn't contain an EFFDT field, the current datetime is used as the key value.

If a SETID is a key on the prompt table, and the record being edited doesn't contain a SETID field, the system looks for SETID on the other records in the current row first, then searches parent rows.

For all other keys, the value used to "select" the correct row in the prompt table record comes only from the record executing the method. No other records (or key field values) are used. You may get unexpected results if not all the keys for the prompt table are filled in (or filled in correctly.)

Considerations for ExecuteEdits and Numeric Fields

A zero (0) might or might not be a valid value for a numeric field. ExecuteEdits processes numeric fields in different ways, depending on whether the field is required:

- If the numeric field is required: 0 is considered invalid.
- If the numeric field is not required: 0 is considered valid.

Parameters

Parameter	*Description*
editlevel	Specifies the standard system edits to be performed against every field on every record. If *editlevel* isn't specified, all system edits are performed. *editlevel* can be any of the following system variables. • %Edit_DateRange • %Edit_OneZero • %Edit_PromptTable • %Edit_Required • %Edit_TranslateTable • %Edit_YesNo

Returns

None.

Example

The following is an example of a call to execute Required Field and Prompt Table edits:

```
&REC.ExecuteEdits(%Edit_Required + %Edit_PromptTable);
```

The following is an example showing how ExecuteEdits() could be used:

```
&REC.ExecuteEdits();
If &REC.IsEditError Then
   LogError();  /*application specific call */
End-If;
```

See Also

Chapter 34, "Record Class," SetEditTable, page 2100; Chapter 34, "Record Class," IsEditError, page 2106; Chapter 18, "Field Class," EditError, page 951; Chapter 18, "Field Class," MessageNumber, page 962 and Chapter 18, "Field Class," MessageSetNumber, page 963

GetField

Syntax

```
GetField({n | FIELD.fieldname})
```

Description

The GetField method instantiates a field object for the specified field associated with the record. This is the default method for the record object. This means that any record object, followed by a parameter list, acts as if GetField is specified.

Note. If the field you're accessing has the same name as a record property (that is, Name or FieldCount) you can't use the default method for accessing the field. You must specify GetField.

For example, the following is invalid for accessing the value of a field called NAME:

```
&NUMBER = &REC.NAME.Value;
```

You must use the following code to get the value of a field called NAME:

```
&NUMBER= &REC.GetField(FIELD.NAME).Value;
```

Parameters

Parameter	*Description*
n | **FIELD.***fieldname*	Specify a field to be used for instantiating the field object. You can specify either *n* or **FIELD.***fieldname.* Specifying *n* creates a field object for the *nth* field in the record. This might be used if writing code that needs to examine all fields in a record without being aware of the name. Specifying **FIELD.***fieldname* creates a field object for the field *fieldname.*

Note. There is no way to predict the order the fields will be accessed. Use the *n* option only if the order in which the fields are processed doesn't matter.

Returns

A field object.

Example

```
&REC.GetField(FIELD.CHARACTER).Value = "Hello";
```

As GetField is the default method for a record object, the following code is identical to the previous code:

```
&REC.CHARACTER.VALUE = "Hello";
```


The following code creates an array containing all the names of all the fields in a related language record.

```
Local array of string &FIELD_LIST_ARRAY;
&FIELD_LIST_ARRAY = CreateArray();
For &I = 1 to &REC_RELATED_LANG.FieldCount
   &FIELD_LIST_ARRAY.Push(&REC_RELATED_LANG.GetField(&I).Name);
End-For;
```

The GetField method requires either FIELD.fieldname or a number. It won't take a string field name. However, you can use the @ operator to convert the string field name into a **FIELD.***fieldname* reference, as follows:

```
&REC = GetRecord();
&REC2 = GetLevel0(1).EMPL_CHECKLIST;
&FIELD2 = &REC.GetField(@ "FIELD." | &REC2.getfield(&I).Name);
```

The following code converts field name strings (using the @ symbol) to component names to get the value of all the fields in a subrecord. Note the code in **Keyword style.**

```
Function get_draft_dst_codes
   /* Load dst id codes into record structures */
   &DSTCODES = CreateRecord(RECORD.DR_DST_CODE_SBR);
   &DRAFTITEM = GetRecord(RECORD.DRAFT_ITEM);
   &DRAFTITEM.CopyFieldsTo(&DSTCODES);
   /* If any missing get from AR dist code, then Draft Type-BU, then BU */
   For &I = 1 To &DSTCODES.FieldCount
      If None(&DSTCODES.GetField(&I).Value) Then
         get_ar_dst_code();
         &NAME = &DSTCODES.GetField(&I).Name;
         If All(&DST.GetField(@ ("FIELD." | &NAME)).Value) Then
            &DSTCODES.GetField(&I).Value = &DST.GetField(@("FIELD." |⇒
 &NAME)).Value;
         Else
            If All(&R_DRAFTBU.GetField(@("FIELD." | &NAME)).Value) Then
               &DSTCODES.GetField(&I).Value = &R_DRAFTBU.GetField(@("FIELD." |⇒
 &NAME)).Value;
            Else
               &DSTCODES.GetField(&I).Value = &R_ARBU.GetField(@("FIELD." |⇒
 &NAME)).Value;

            End-If;
         End-If;
      End-If;
   End-For;
   /* Copy the defaulted values back to draft item */
   &DSTCODES.CopyFieldsTo(&DRAFTITEM);
End-Function;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetField

Insert

Syntax

Insert()

Description

The Insert method uses the field names of the record and their values to build and execute an Insert SQL statement which adds the given record (row of data) to the SQL table.

Because this method results in a database change, it can only be issued in the following events:

- SavePreChange
- WorkFlow
- SavePostChange

If your application is repeating the same instruction many times, such as doing a million INSERTs, use the SQL object with the BulkMode property set to True, rather than the record SQL methods.

If you're using a record created using CreateRecord, all fields are initially set to "", 0, or NULL, depending on the type of field. If you don't specify values for these fields, these initial values are written to the database when the insert is executed. If you want the default values for the fields inserted instead, use the SetDefault method prior to using the Insert method.

For every record inserted with the Insert method, if the set language is not the base language and the record has related language records, the Insert method tries to do related language processing.

See Also

Chapter 40, "SQL Class," page 2239; *PeopleTools 8.51 PeopleBook: Global Technology*, "Using Related Language Tables" and Chapter 34, "Record Class," SetDefault, page 2099

Parameters

None.

Returns

The result is True on successful completion, False if there was a record with those keys already in the database, that is, if a duplicate record was found. Any other conditions cause termination.

Example

Suppose that KEYF1 and KEYF2 are the key fields of record definition MYRECORD, and that this record definition also contains the field definitions MYRF3, MYRF4. The following code adds a MYRECORD record (row of data) to the database, with KEYF1 set to "A", KEYF2 set to "B", MYRF3 set to "X", and MYRF4 set to "Y":

```
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
&REC.Insert();
```


See Also

Chapter 34, "Record Class," Delete, page 2085; Chapter 34, "Record Class," SelectByKey, page 2096 and Chapter 34, "Record Class," Update, page 2102

Save

Syntax

```
Save([CopyToOriginal])
```

Description

The Save method saves the record to the database in two steps.

1. Check to see that it is safe to save this record to the database.
2. Either insert this record or update an existing record.

To accomplish the first step and determine that it is safe to save the record, this method uses similar logic to that used in the component processor when it saves a record. To do that a "select for update" SQL statement is executed. If, as a result of this statement, the record does not exist in the database, this method simply inserts the record into the database. All the processing that occurs for the insert method then occurs.

If the result of the "select for update" statement returns a record, the values of all the fields in the database record are compared to the original values of the record executing this method. If they are equal, indicating that no other user has updated the record, the record in the database is updated. All the processing that occurs for the update method then occurs. If the result of the above comparison fails, indicating that there has been an update of the database record in the meantime, perhaps by some other user, the save method fails and returns a false result.

For the update process to succeed, the record object must contain both original and changed values of the data. It is crucial to get data loaded into the record object correctly. To do this, do one of the following:

- Derive the record object from the component buffers.
- Derive the record object from a rowset filled using the Fill method.
- Load the data using the record class SelectByKey and SelectByKeyEffdt methods specifying the optional Boolean parameter to be False.

See the note and example below.

Of course when you use Save to simply insert data you do not need load data into the record object this way.

Use this method when you are writing applications that do not use the facilities that come predelivered with the component processor and you must deal with issues of data contention.

For example, suppose you are writing an application that deals with course registration. You have to handle changing the number of students who are registered carefully if it is possible for more than one user to be registering for the same course at the same time. The normal way is to use the component processor to handle data contention by placing the record that deals with the number of students registering on some page. Then if two users try to update the same record one of them receives a message indicating that the page has been updated by another user.

If you write an application, outside the component processor, that relies on the record class to manage the number of students enrolled, it is not sufficient to use the record class Insert or Update methods to maintain data integrity. Suppose one user registers for the course and sees there are 10 other registrants. At the same time another user registers for the course and sees that there are also 10 registrants. Suppose the maximum is 11 students. Both users at this point believe they can register since at the point they accessed the data from the database there were only 10 students. So far, so good. The trouble comes when they save their registrations. If the application uses the update method, after simply adding one to the number of registrants, both updates will succeed indicating there are now 11 students in the course when in fact there are now 12. It is for cases such as these that the Save method applies since it guarantees that one of the save calls will fail. It is up to the application then to let the user know that while they were thinking about it someone else registered for the course and took the last spot.

The Save method is important to maintain data integrity. However, it can execute more slowly than saving within the component processor.

Since this method results in a database change, and does either insert or update processing, all the considerations and implications of using the Insert or Update methods apply. See the description of those methods for more detail.

The parameter *CopyToOriginal* indicates what action should occur on a successful save. If this parameter is true, the system copies the current changed buffers for the record into the original buffers for the record. This means that subsequent Saves, without reloading data from the database will run correctly. However if there are database specific updates to this record occurring outside of this application's control (such as, using database specific triggers,) it is always advisable to reload the data from the database.

The default value for this optional parameter is false, indicating that no update of the original buffers for this record occurs. If you are certain that there are no database updates occuring outside of your application's on save processing, you can specify this optional parameter to be true and save the overhead of having to reload the data for this record from the database. If you are unsure you should always reload the record data after calling the Save method.

Note. This method does *not* update fields marked as System maintained in the record definition in Application Designer.

Since this save is done outside the component processor save processing, no save processing PeopleCode that is associated with this record is executed.

Use of this method that results in an update, and requires that the data in the record object is such that both a before and after image of the data is kept. When the system loads data into the component buffers there is a concept of the "original data" (data loaded from the database) and "changed data" (data changed by the user or the application). This allows the system to determine when saving, whether the data has been changed by another user. If you use this method with record objects not derived from the component buffers you need to make sure that the data in the record object has both a before and after image otherwise the save method fails. Record objects derived from rowsets that have been populated using the Fill method will not produce an error. However if you load data into a record object using either the SelectByKey or SelectByKeyEffdt record class methods, you need to specify the *CopyToOriginal* parameter so the system will load the data into the original data buffers, not the changed data buffers. To do that specify the optional parameter to be False. See the example below.

See Also

Chapter 34, "Record Class," Insert, page 2090 and Chapter 34, "Record Class," Update, page 2102

PeopleTools 8.51 PeopleBook: Global Technology, "Using Related Language Tables"

Parameters

Parameter	*Description*
CopyToOriginal	Specify what action should occur on a successful save. This parameter takes a Boolean value: true, then the system copies the current changed buffers for the record into the original buffers for the record, false, the buffers are not copied. The default value is false.

Returns

The result is True on successful completion, False if there were no changed fields in the record or it was not safe to save this record (see above for details). Any other conditions cause termination.

Note. It is not possible to ignore the return code from the Save method in your PeopleCode.

Example

Suppose that KEYF1 and KEYF2 are the key fields of record definition MYRECORD, and that this record definition also contains the record field definitions MYRF3, MYRF4. The following code tries to save MYRECORD record in the database with KEYF1 set to "A", KEYF2 set to "B", setting MYRF3 to "X" and MYRF4 to "Y". This example works with a brand new record object and results in an insert if successful.

```
/* First -- create a brand new record */
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";

/* Second -- update values as needed */
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
/* Finally save it */
if &REC.Save() then
    /* handle the successful return */
else
    /* handle the unsuccessful return message to user? Try again? */
end-if;
```

This example works with an existing record and results in an update if successful.


```
/* First - create the record and get the data from the db */
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";
&REC.SelectByKey(False);
/* False retrieves data into the original, not changed buffers. */

/* Second - update values as needed */
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
/* Finally save it */
if &REC.Save() then
    /* handle the successful return */
else
    /* handle the unsuccessful return - message to user? Try again? */
end-if;
```

See Also

Chapter 34, "Record Class," Insert, page 2090; Chapter 34, "Record Class," Delete, page 2085; Chapter 34, "Record Class," SelectByKey, page 2096 and Chapter 34, "Record Class," Update, page 2102

Chapter 34, "Record Class," SelectByKeyEffDt, page 2098

SearchClear

Syntax

SearchClear()

Description

The SearchClear method clears the field values for all search keys for a record.

Considerations Using SearchClear and SearchDefault

The field property SearchDefault sets a field to its default value (if there is one) immediately after SearchInit PeopleCode finishes. SearchDefault overrides SearchClear. If you call SearchClear for a record, then use SearchDefault for a field, the field is set to its default value and the search key values for the rest of the record are cleared.

Parameters

None.

Returns

None.

Example

```
Local Record &REC;

&REC = GetRecord();
&REC.SearchClear();
```

See Also

Chapter 18, "Field Class," SearchDefault, page 966 and Chapter 18, "Field Class," SearchClear, page 944

SelectByKey

Syntax

SelectByKey([*UseChangedBuffers*])

Description

The SelectByKey method uses the key field names of the record and their values to build and execute a Select SQL statement. The field values are then fetched from the database SQL table into the record object executing the method, and the Select statement is closed.

Note. You can't use this method in dynamic views.

If you don't specify all the key fields for a record, those you exclude are added to the Where clause with the condition equal to a blank value. If not all keys are set and more than one row is retrieved, you won't receive an error and SelectByKey won't fetch any data.

SelectByKey returns a single row of data. To fetch more than one row, use the SQL object.

For every record read by the SelectByKey method, if the set language is not the base language and the record has related language records, the SelectByKey method tries to do related language processing.

You can also select into a record using SQLExec and meta-SQL. For example, if you need to use SelectByKey with an effective date, you can use the meta-SQL %SelectByKeyEffDt in a SQLExec statement. In the following example, &RECOBJ is the name of a record created using the CreateRecord function.

```
SQLExec("%SelectByKeyEffDt(:1, :2)", &RECOBJ, %Date, &RECOBJ);
```

See Also

Chapter 40, "SQL Class," page 2239

PeopleTools 8.51 PeopleBook: Global Technology, "Using Related Language Tables"

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SQLExec

Parameters

Parameter	*Description*
UseChangedBuffers	Specify whether to retrieve the data into the changed or original buffers. The system keeps track of the original and changed values. This parameter takes a Boolean value: true, retrieve into the changed buffers, False, use the original buffers. The default value is true. is to retrieve the data into the changed buffers. To use this parameter with the Save method, it is important that data be loaded into the original buffers in order for the save logic to work. To do that specify a value of false for this parameter. See Chapter 34, "Record Class," Save, page 2092.

Returns

The result is true on successful completion, false if the record was not found. Any other conditions cause termination.

If false is returned, the system resets the record values to their default values for scrolls that are not derived. If you do a manual insert afterwards (using the Insert method) you may insert a record with null keys.

Example

Suppose that KEYF1 and KEYF2 are the two key fields of record definition MYRECORD. The following code reads the database record that has KEYF1 equal to "A" and KEYF2 equal to "X" into the &REC record object:

```
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "X";
&REC.SelectByKey();
```

The following example verifies if the method completes successfully before using the Insert method.

```
if not &Rec.SelectByKey() then

     /* Must duplicate this key field population before the Insert - else can get⇒
 'blank' row */

      &Rec.KEY1.Value = ...;
      &Rec.KEY2.Value = ....;
      ....

     &Rec.Insert();

end-if;
```

See Also

Chapter 34, "Record Class," Delete, page 2085; Chapter 34, "Record Class," Insert, page 2090; Chapter 34, "Record Class," Update, page 2102 and Chapter 34, "Record Class," Save, page 2092

SelectByKeyEffDt

Syntax

`SelectByKeyEffDt(Date[, UseChangedBuffers])`

Description

Use the SelectKeyByEffDt to build and execute a SQL Select statement to get the current effective row based on an "as of date". The field values are then fetched from the database SQL table into the record object executing the method, and the Select statement is closed.

Note. You can't use this method in dynamic views.

SelectByKeyEffDt returns a single row of data. To fetch more than one row, use the SQL object.

For every record read by the SelectByKeyEffDt method, if the set language is not the base language and the record has related language records, the SelectByKeyEffDt method tries to do related language processing. The system assumes that both the base table and the related language table are effective-dated and that the effective-date keys are the same in each.

To use other keys, more than just the effective date, use the SelectByKey Record class method.

Parameters

Parameter	***Description***
Date	Specify the "as of" date you want to use as the effective date for the Select statement.
UseChangedBuffer	Specify whether to retrieve the data into the changed or original buffers. The system keeps track of the original and changed values. This parameter takes a Boolean value: true, retrieve into the changed buffers, False, use the original buffers. The default value is true. is to retrieve the data into the changed buffers. To use this parameter with the Save method, it is important that data be loaded into the original buffers in order for the save logic to work. To do that specify a value of false for this parameter. See Chapter 34, "Record Class," Save, page 2092.

Returns

The result is True on successful completion, False if the record was not found. Any other conditions cause termination.

If false is returned, the system resets the record values to their default values for scrolls that are not derived. If you do a manul insert afterwards (using the Insert method) you may insert a record with null keys.

Example

```
Local Record &Rec;
Local any &Date;

&Date = %Date;
&Rec = CreateRecord(Record.COMPANY_TBL);
&Rec.COMPANY.Value = "CCB";

If &Rec.SelectByKeyEffDt(&Date) Then
   &Out = &Rec.DESCR.Value | " as of " | &Rec.EFFDT.Value | "->" |⇒
 &Rec.STREET1.Value;
   WinMessage("Found this company: " | &Out);
Else
   WinMessage("Not there!");
End-If;
```

See Also

Chapter 34, "Record Class," Delete, page 2085; Chapter 34, "Record Class," Insert, page 2090; Chapter 34, "Record Class," Update, page 2102; Chapter 34, "Record Class," Save, page 2092 and Chapter 34, "Record Class," SelectByKey, page 2096

Chapter 40, "SQL Class," page 2239

PeopleTools 8.51 PeopleBook: Global Technology, "Using Related Language Tables"

SetDefault

Syntax

SetDefault()

Description

SetDefault sets the value of every field in the record to a null value, or to a default, depending on the type of field.

- If this method is used against data from the Component buffers, the next time default processing occurs, it is set to its default value: either a default specified in its record field definition or one set programmatically by PeopleCode located in a FieldDefault event. If neither of these defaults exist, the Component Processor leaves the field blank.
- If this method is used with a field that isn't part of the data buffer (for example, a field in a record object instantiated with CreateRecord) the field is automatically set to its default value if one is set for the field, not for the record field. Any FieldDefault PeopleCode does not run on these types of fields. To set the default values for just one field, use the SetDefault field class method.

Blank numbers correspond to zero on the database. Blank characters correspond to a space on the database. Blank dates and long characters correspond to NULL on the database. SetDefault gives each field data type its proper value.

Parameters

None.

Returns

None.

Example

```
&CHARACTER.SetDefault();
```

See Also

Chapter 18, "Field Class," SetDefault, page 946

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Default Processing

SetEditTable

Syntax

`SetEditTable``(`*`%PromptField,`*` RECORD.`*`recordname`*`)`

Description

The SetEditTable method works with the ExecuteEdits method. It sets the value of a field on a record that has its prompt table defined as *%PromptField* value. %PromptField values are used to dynamically change the prompt record for a field.

There are several steps to setting up a field so that you can dynamically change its prompt table.

To set up a field with a dynamic prompt table:

1. Define a field in the DERIVED record called *fieldname.*

2. In the record definition for the field you want to have a dynamic prompt table, define the prompt table for the field as %*PromptField,* where *PromptField* is the name of the field you created in the DERIVED record.

   Note. When you use SetEditTable, you don't have to add a hidden field to the page.

3. Use SetEditTable to dynamically set the prompt table.

%PromptField is the name of the field on the DERIVED work record. RECORD.*recordname* is the name of the record to be used as the prompt table.


```
&REC.SetEditTable("%EDITTABLE1", Record.DEPARTMENT);
&REC.SetEditTable("%EDITTABLE2", Record.JOB_ID);
&REC.SetEditTable("%EDITTABLE3", Record.EMPL_DATA);
&REC.ExecuteEdits();
```

Every field on a record that has the prompt table field %EDITTABLE1 will have the same prompt table, that is, DEPARTMENT.

See Chapter 34, "Record Class," ExecuteEdits, page 2086 and *PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide*, "Creating Record Definitions."

Considerations for SetEditTable and ExecuteEdits

The keys used to "select" the correct row in the prompt table record come only from the record object executing the method. All the keys from the component are not used. You may get unexpected results if not all the keys for the prompt table are filled in (or filled in correctly.)

Parameters

Parameter	***Description***
%PromptField	Specifies the name of the field in the DERIVED record that has been defined as the prompt table for a field in the record definition.
RECORD.*recordname*	Specifies the name of the record definition that contains the correct standard system edits to be performed for that field in the record.

Returns

None.

Example

```
/* To Set the Prompt Table */
&Der_EditRec = &TSKPRF_Det_Row.GetRecord(Record.DERIVED);
&Prompt_Editname = &Der_EditRec.EDITTABLE.Value;
&Prompt_Edittable = "Record." | &Prompt_Editname;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE", @&Prompt_Edittable);

&Prompt_Editname2 = &Der_EditRec.EDITTABLE2.Value;
&Prompt_Edittable2 = "Record." | &Prompt_Editname2;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE2", @&Prompt_Edittable2);

&Prompt_Editname3 = &Der_EditRec.EDITTABLE3.Value;
&Prompt_Edittable3 = "Record." | &Prompt_Editname3;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE3", @&Prompt_Edittable3);

&Prompt_Editname4 = &Der_EditRec.EDITTABLE4.Value;
&Prompt_Edittable4 = "Record." | &Prompt_Editname4;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE4", @&Prompt_Edittable4);

&Prompt_Editname5 = &Der_EditRec.EDITTABLE5.Value;
&Prompt_Edittable5 = "Record." | &Prompt_Editname5;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE5", @&Prompt_Edittable5);

&Prompt_Editname6 = &Der_EditRec.EDITTABLE6.Value;
&Prompt_Edittable6 = "Record." | &Prompt_Editname6;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE6", @&Prompt_Edittable6);

&TSKPRF_Det_Rec.ExecuteEdits(%Edit_Required + %Edit_PromptTable);
```

See Also

Chapter 34, "Record Class," ExecuteEdits, page 2086; Chapter 34, "Record Class," IsEditError, page 2106; Chapter 18, "Field Class," EditError, page 951; Chapter 18, "Field Class," MessageNumber, page 962 and Chapter 18, "Field Class," MessageSetNumber, page 963

Update

Syntax

Update([*KeyRecord*])

Description

The Update method uses the changed fields of the record and their values to build and execute an Update SQL statement, updating the SQL table. If you are updating the key fields of the record, you must supply a record object *KeyRecord* that contains the old values of the key record fields.

Because this method results in a database change, it can only be issued in the following events:

- SavePreChange
- WorkFlow

- SavePostChange

If your application is repeating the same instruction many times, such as doing a million UPDATEs, use the SQL object with the BulkMode property set to True, rather than the record SQL methods.

For every record updated by the Update method, if the set language is not the base language and the record has related language records, the Update method tries to do related language processing.

Note. This method does not update fields marked as system maintained in Application Designer.

See Also

Chapter 40, "SQL Class," page 2239 and *PeopleTools 8.51 PeopleBook: Global Technology*, "Using Related Language Tables"

Parameters

Parameter	***Description***
KeyRecord	If you're updating the key fields of the record, you must supply the old key field values in the record object *KeyRecord.*

Returns

The result is True on successful completion, False if the record was not found or (when updating the key fields) a duplicate record was found. Any other conditions cause termination.

Example

Suppose that KEYF1 and KEYF2 are the key fields of record definition MYRECORD, and that this record definition also contains the record field definitions MYRF3, MYRF4. The following code updates the MYRECORD record in the database with KEYF1 set to "A", KEYF2 set to "B", setting MYRF3 to "X" and MYRF4 to "Y":

```
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
&REC.Update();
```

This code updates the MYRECORD record in the database with KEYF1 of "A" and KEYF2 of "B", setting KEYF2 to "C", MYRF3 to "M" and MYRF4 to "N":


```
Local record &REC1, &REC2;
&REC1 = CreateRecord(RECORD.MYRECORD);
&REC2 = CreateRecord(RECORD.MYRECORD);
&REC1.KEYF1.Value = "A";
&REC1.KEYF2.Value = "B";
&REC2.KEYF1.Value = "A";
&REC2.KEYF2.Value = "C";
&REC2.MYRF3.Value = "M";
&REC2.MYRF4.Value = "N";
&REC2.Update(&REC1);
```

See Also

Chapter 34, "Record Class," Delete, page 2085; Chapter 34, "Record Class," SelectByKey, page 2096 and Chapter 34, "Record Class," Insert, page 2090

Record Class Properties

In this section, we discuss the Record class properties. The properties are listed in alphabetical order.

FieldCount

Description

This property returns the total number of fields contained in the record. This value is a number.

This property is read-only.

Example

```
WinMessage("This record has this many fields : " | &REC.FieldCount);
```

fieldname

Description

If a field name is used as a property, it accesses the field object with that name. This means the following code:

&REC.*fieldname*

acts the same as

&REC.GetField(FIELD.*fieldname*);

Note. If the field you're accessing has the same name as a record property (such as, Name) you can't use this method for accessing the field. You must use the GetField method.

If you combine the *fieldname* property with the *recname* property, you obtain the specified field object associated with that record from the row.

```
&ROW.recname.fieldname
```

This property is read-only.

Example

```
&REC.CHARACTER.Enabled = True;
```

IsChanged

Description

This property returns True if any field value on the primary database record of the row has been changed.

Note. This property is for use only with the primary database record. It does not return valid results if used with a work record.

This property is read-only.

Considerations Using IsChanged

This property and the IsChanged row property do not always return identical values.

If a row in a scroll contains multiple records (such as a primary database record and one or more work records), the IsChanged row property returns True if any of the records in the row are changed. The IsChanged record property returns True only if a specific record, namely, the primary database record, is changed.

If a row is deleted from a scroll, only the primary database record has its IsChanged property marked to False (since the row has been deleted.) Any work records in the row still have their IsChanged properties set to True.

Example

```
If &REC.IsChanged Then
    Warning("This Record has been changed");
End-if;
```

IsDeleted

Description

This property is True if the record has been deleted. For a level zero record, it is possible for a record to be deleted without the whole row being deleted. For other levels, this property is the same as the row property IsDeleted.

This property is read-only.

Example

```
&tmp = &REC.IsDeleted;
```

See Also

Chapter 35, "Row Class," IsDeleted, page 2121

IsEditError

Description

This property is True if an error has been found for any field associated with the record after executing the ExecuteEdits method. This property can be used with the Field class properties EditError (to find out which field is in error), and MessageSetNumber and MessageNumber to find the error message set number and error message number.

The IsEditError property returns True when a field with a Null value is encountered.

This property is read-only.

Example

The following is an example showing how IsEditError, along with the ExecuteEdits method could be used:

```
&REC.ExecuteEdits();
If &REC.IsEditError Then
 For &I = 1 to &REC.FieldCount
   If &REC.GetField(&I).EditError Then
      LOG_ERROR();  /* application specific call */
   End-If;
 End-For;
End-If;
```

See Also

Chapter 34, "Record Class," ExecuteEdits, page 2086; Chapter 34, "Record Class," IsEditError, page 2106; Chapter 18, "Field Class," EditError, page 951; Chapter 18, "Field Class," MessageNumber, page 962 and Chapter 18, "Field Class," MessageSetNumber, page 963

Name

Description

This property returns the name of the record definition of the record as a string.

To access an SQL table name for a record, use the %Table meta-SQL construct.

This property is read-only.

Example

```
WinMessage("The name of this record is : " | &REC.Name);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "Meta-SQL Elements," %Table

ParentRow

Description

This property returns the row object for the row containing the record. If the record object was created with the CreateRecord built-in function, the value for ParentRow is null because the record object isn't part of a row.

This property is read-only.

Example

```
&ROWOBJECT = &REC.ParentRow;
    &TMP = "The row number of the parent row is " | &ROWOBJECT.RowNumber;
/* note that RowNumber is a property of the row class  */
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRecord

RelLangRecName

Description

This property returns the name of the related language record as a string. The value will be a null string ("") if there is no related language record.

Example

```
Local Record &REC;

&REC = GetRecord();
&RECNAME_BASE = &REC.Name;
&RELLANGRECNAME = &REC.RelLangRecName;
SQLExec("select RELLANGRECNAME from PSRECDEFN where RECNAME = :1", &RECNAME_BASE,⇒
 &RELLANGRECNAME);
```

SystemIDFieldName

Description

Note. PeopleSoft Mobile Agent is a deprecated product. This mobile property currently exists for backward compatibility only.

This property returns the name of the System ID field as a string. If there is no defined value (the default) the value is a null string ("").

This property accesses the System ID field name value specified on the Use tab of the Record Properties dialog box.

The System ID field is used to create a unique way to identify the record for mobile synchronization purposes. This property is used exclusively for mobile applications. This property is not available for subrecords.

This property is read-only.

Example

```
Local Record &REC;

&REC = GetRecord();
&sSystemID = &REC.SystemIDFieldName;
```

See Also

Chapter 34, "Record Class," TimeStampFieldName, page 2109

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Record Definitions," Setting Record Properties

TimeStampFieldName

Description

Note. PeopleSoft Mobile Agent is a deprecated product. This mobile property currently exists for backward compatibility only.

This property returns the name of the Timestamp field as a string. If there is no defined value (the default) the value is a null string ("").

This property accesses the Timestamp field name value specified on the Use tab of the Record Properties dialog box.

The Timestamp field is used to specify a field that is automatically updated with the date and time when there's a change to the record. This property is used exclusively for mobile applications.

This property is not available for subrecords. This property is read-only.

Example

```
Local Record &REC;

&REC = GetRecord();
&sTimeStamp = &REC.TimeStampFieldName;
```

See Also

Chapter 34, "Record Class," SystemIDFieldName, page 2108

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Record Definitions," Setting Record Properties

Chapter 35

Row Class

This chapter provides an overview of Row class and discusses the following topics:

- Shortcut considerations.
- Data type of a row object.
- Scope of a row object.
- Row class built-in function.
- Row class methods.
- Row class properties.

Understanding Row Class

A *row* object, instantiated from the Row class, is a single row of data that consists of one to *n* records of data. A single row in a component scroll is a row.

A row object is always associated with a rowset object, that is, you cannot instantiate a row object without first either explicitly or implicitly instantiating a rowset object.

A row may have one to *n* child rowsets. For example, a row in a level two scroll may have *n* level three child rowsets.

CopyTo and GetRecord are two commonly use methods for this class. Visible and IsChanged are two commonly used properties for this class.

The row class is one of the data buffer access classes.

If a rowset object is instantiated using the CreateRowset function, the rowset object that's instantiated is a standalone rowset. Delete and insert activity on these types of rowsets aren't automatically applied at save time. Use standalone rowsets for work records.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

Shortcut Considerations

The default method for the row class is GetRecord. This means you can specify just a record name to access a record on the row. For example, the following two lines of code are equivalent:

```
&Count = &MyRow.EMPL_CHECKLIST.FieldCount;
```

```
&Count = &MyRow.GetRecord(RECORD.EMPL_CHECKLIST).FieldCount;
```

In addition, the row class has a method *scrollname.* This enables you to access a specific child rowset *and* row within that rowset. This isn't a default method: you're not accessing a child object, but rather, something within that object. For example, the following two lines of code are equivalent:

```
&ChildRow = &MyRow.EMPL_CHKLST_ITM(&I);
```

```
&ChildRow = &MyRow.GetRowset(SCROLL.EMPL_CHKLST_ITM).GetRow(&I);
```

Data Type of a Row Object

Row objects are declared as type Row. For example,

```
Local Row &MYROW;
```

Scope of a Row Object

A Row can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, Component Interface PeopleCode, and so on.

Row Class Built-in Function

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRow

Row Class Methods

In this section, we discuss each Row class method. The methods are discussed in alphabetical order.

CopyTo

Syntax

CopyTo*(row)*

Description

The CopyTo method copies *from*the row executing the method *to*the specified target row, copying *like-named* record fields and subscrolls at corresponding levels. The target row object must be from a "related" rowset, that is, built from the same records, but it can be under a different parent at higher levels.

This method copies *all* the records from the row object executing the method to the target row, not just the primary data record associated with the rowset.

Parameters

Parameter	***Description***
row	Specify a target row. This must be a row object, not a row number.

Returns

None

Example

The following example copies rows from a child rowset (DERIVED_HR) to the current rows.

```
Local Row &ROW;
Local Rowset &RS1;

&RS1 = GetRowset();
For &I = 1 to &RS1.ActiveRowCount
   &ROW = GetRowset(SCROLL.DERIVED_HR).GetRow(&I);
   &ROW.CopyTo(GetRow(&I));
End-For;
```

See Also

Chapter 35, "Row Class," ParentRowset, page 2122; Chapter 34, "Record Class," CopyFieldsTo, page 2082 and Chapter 34, "Record Class," CopyChangedFieldsTo, page 2081

GetNextEffRow

Syntax

GetNextEffRow()

Description

The GetNextEffRow method finds the row, in the same rowset as the row executing the method, that has the next effective date (when compared to the row executing the method.)

Parameters

None

Returns

A row object. If there is no row with a later effective date, this method returns a null object.

Example

```
&ROW2 = &ROW.GetNextEff();

If &ROW2 <> NULL Then
   &TMP = &ROW2.RowNumber;
   /* other processing */
Else
   /* no effective dated row - do error processing */
End-if;
```

See Also

Chapter 35, "Row Class," GetPriorEffRow, page 2114; Chapter 36, "Rowset Class," GetCurrEffRow, page 2143 and Chapter 36, "Rowset Class," DeleteEnabled, page 2163

GetPriorEffRow

Syntax

GetPriorEffRow()

Description

The GetPriorEffRow method finds the row, from the same rowset as the row executing the method, that has the prior effective date to the row executing the method.

Parameters

None

Returns

A row object. If there is no row with a prior effective date, this method returns a null object.

Example

```
&TMP = &ROW.GetPriorEffrow().RowNumber;
```

See Also

Chapter 35, "Row Class," GetNextEffRow, page 2114; Chapter 36, "Rowset Class," GetCurrEffRow, page 2143 and Chapter 36, "Rowset Class," DeleteEnabled, page 2163

GetRecord

Syntax

GetRecord({*n* | RECORD.*recname*})

Description

The GetRecord method creates a record object that references the specified record within the current row object. This is the *default method* for a row object. This means that any row object, followed by a parameter list, acts as if GetRecord is specified.

Parameters

Parameter	*Description*
n | RECORD.*recname*	Specify a record to be used for instantiating the record object. You can specify either *n* or RECORD.*recname*, Specifying *n* creates a record object for the *nth* record in the row. This might be used if writing code that needs to examine all records in a row without being aware of the record name. Specifying RECORD.*recname* creates a record object for the record *recname*.

Note. There is no way to predict the order the records will be accessed. Use the *n* option only if the order in which the records are processed doesn't matter.

Returns

A record object.

Example

The following example gets a record, then assigns the name of the record to the variable &TMP:

```
&REC = &ROW.GetRecord(RECORD.QEPC_LEVEL1_REC);

&TMP = &REC.NAME;  /* note that NAME is a property of the record class */
```

Because GetRecord is the default method for a row, the following code is identical to the previous:

```
&REC = &ROW.QEPC_LEVEL1_REC;

&TMP = &REC.NAME;
```

The following example loops through all the records for a row:

```
Local rowset &LEVEL1;
Local row &ROW;
Local record &REC;

&LEVEL1 = GetRowset(SCROLL.EMPL_CHECKLIST);
For &I = 1 to &LEVEL1.ActiveRowCount
   &ROW = &LEVEL1.GetRow(&I);
   For &J = 1 to &ROW.RecordCount
      &REC = &ROW.GetRecord(&J);
         /* do processing */
   End-For;
End-For;
```

The following function takes a rowset and a record, passed in from another program. GetRecord won't take a variable for the record, however, using the @ symbol you can convert the string to a component record name.

```
Function Get_My_Row(&PASSED_ROWSET, &PASSED_RECORD)

   For &ROWSET_ROW = 1 To &PASSED_ROWSET.RowCount
      &UNDERLYINGREC = "RECORD." | &PASSED_ROWSET.DBRecordName;
      &ROW_RECORD = &PASSED_ROWSET.GetRow(&ROWSET_ROW).GetRecord(@&UNDERLYINGREC);

      /* Do other processing */

   End-For;

End-Function;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRecord and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," CreateRecord

GetRowset

Syntax

```
GetRowset({n | SCROLL.scrollname})
```

Description

The GetRowset method creates a rowset object that references a child rowset of the current row. *scrollname* must specify the primary record for the child rowset.

Parameters

Parameter	*Description*
n | **SCROLL.***scrollname*	Specify a rowset to be used for instantiating the rowset object. You can specify either *n* or SCROLL.*scrollname.* Specifying *n c*reates a rowset object for the *nth* child rowset in the row. This might be used if you are writing code that examines all rowsets in a row without being aware of the name. Specifying SCROLL. *scrollname* creates a rowset object for the record *scrollname.scrollname* must specify the primary record for the child rowset.

Note. There is no way to predict the order the rowsets will be accessed. Use the *n* option only if the order in which the rowsets are processed doesn't matter.

Returns

A rowset object.

Example

```
&CHILDROWSET = &ROW.GetRowset(SCROLL.QEPC_LEVEL1_REC);
```

See Also

Chapter 35, "Row Class," scrollname, page 2118

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

scrollname

Syntax

```
scrollname(n)
```

Description

If *scrollname* is used as a method name for a row, it is used to select a child rowset and a row within that rowset.

```
&ROW.scrollname(n)
```

acts the same as

```
&ROW.GetRowset(SCROLL.scrollname).GetRow(n)
```

Parameters

Parameter	*Description*
n	Must be a valid row number for the selected child rowset.

Returns

A row object.

Example

```
&SUBROW1 = &ROW.QEPC_LEVEL1_REC(1);
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

Row Class Properties

In this section, we discuss the Row class properties. The properties are discussed in alphabetical order.

ChildCount

Description

This property returns the number of child rowsets of the row. It is defined by the "structure" of the scroll, so it is the same for all rows of the rowset.

It might be used, in conjunction with the GetRowset method, to write code that examines all child rowsets.

This property is read-only.

Example

```
For &I = 1 to &ROW.ChildCount
   &ROWSET = &ROW.GetRowset(&I);
      /* do processing */
End-For;
```

DeleteEnabled

Description

This property determines whether a row can be deleted (the equivalent of the user pressing ALT+8 and ENTER). This property takes a Boolean value.

Note. This property controls only whether an end-user can delete a row. Rows can still be deleted using PeopleCode.

This property is typically used to prevent deletion of an individual row that the other properties controlling deletion would allow to be deleted. The following table goes through the Boolean logic.

Will delete be allowed by user?	***Others enable***	***Others disable***
DeleteEnabled = True	Yes	No
DeleteEnabled = False	No	No

The initial value of this property is True.

Note. If No Row Delete is selected in Application Designer, setting the DeleteEnabled row property to True will *not* override this value. If you want to control whether a row can be deleted at runtime, you should *not* select No Row Delete at design time. Likewise, if the DeleteEnabled Rowset property is False, setting the DeleteEnabled Row property to True will *not* override the rowset property. If you want to control whether an individual row can be deleted at runtime, you should not set the DeleteEnabled rowset property to False.

For consistency, PeopleSoft recommends that either all rows at a level should disable deletions, or they should all allow deletions.

For rows created with non-Component Processor data (such as message rowsets, Application Engine rowsets, and so on) this property has no effect.

Note. Don't use this property with rows from rowsets created using CreateRowset. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren't automatically applied at save time.

This property is read-write.

See Also

Chapter 36, "Rowset Class," DeleteEnabled, page 2163

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Using Scroll Areas, Scroll Bars, and Grids," Using Scroll Areas and Scroll Bars

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

IsChanged

Description

This property returns True if any field value on the primary database record of the row has been changed.

Note. Use this property only with the primary database record. It doesn't return valid results if used with a work record. This property returns True only if the existing row object has a field that has changed. If a field in a lower level rowset has changed, this property returns False.

This property is read-only.

Considerations Using IsChanged

This property and the IsChanged record property do *not*always return identical values.

If a row in a scroll contains multiple records (such as a primary database record and one or more work records), the IsChanged row property returns True if *any* of the records in the row are changed. The IsChanged record property returns True *only* if a specific record, namely, the primary database record, is changed.

If a row is deleted from a scroll, *only*the primary database record has its IsChanged property marked to False (since the row has been deleted.) Any work records in the row *still*have their IsChanged properties set to True.

Example

```
&tmp =  &ROW.IsChanged;
if &tmp = True then
   Warning("A Field on this row has been changed");
End-If;
```

IsDeleted

Description

This property returns True if the row has been deleted.

This property is read-only.

Note. This property also returns True if the row is new and hasn't been changed.

Example

```
&tmp = &ROW.IsDeleted;
```

IsEditError

Description

This property is True if an error has been found for any field associated with any record in the current row or on any row in any child rowset of the current row after executing the ExecuteEdits method. This property can be used with the Field class properties EditError (to find out which field is in error), and MessageSetNumber and MessageNumber to find the error message set number and error message number.

This property is read-only.

Example

The following is an example showing how IsEditError, along with the ExecuteEdits method could be used:


```
&MSG.ExecuteEdits();
If &MSG.IsEditError Then
&RS = &MSG.GetRowset();
   For &J = 1 to &RS.RowCount
      &ROW = &RS.GetRow(&J);
         If &ROW.IsEditError Then
            &REC = &ROW.GetRecord();
            For &I = 1 to &REC.FieldCount
               If &REC.GetField(&I).EditError Then
                  LOG_ERROR();  /* application specific call */
               End-If;
            End-For;
      End-If;
   End-For;
End-If;
```

See Also

Chapter 34, "Record Class," ExecuteEdits, page 2086; Chapter 34, "Record Class," IsEditError, page 2106; Chapter 18, "Field Class," EditError, page 951; Chapter 18, "Field Class," MessageNumber, page 962 and Chapter 18, "Field Class," MessageSetNumber, page 963

IsNew

Description

This property is True if the row is a new (inserted) row.

This property is read-only.

Example

```
&tmp = &ROW.IsNew;
```

ParentRowset

Description

This property returns a rowset object referencing the rowset containing the row.

This property is read-only.

Example

```
&tmp = &ROW.ParentRowset.RowCount;

/* note that rowcount is a property of the Rowset class */
```

recname

Description

If a record name is used as a property, it accesses the record object with that name. This means the following code:

```
&ROW.somerecname
```

acts the same as

```
&ROW.GetRecord(RECORD.somerecname);
```

This property is read-only.

Example

```
&REC = &ROW.QEPC_LEVEL1_REC;
```

RecordCount

Description

This property returns the number of records in the row. It is defined by the "structure" of the scroll, so it is the same for all rows of the rowset.

This property might be used in conjunction with the GetRecord method to write code that examines all records in some way.

This property is read-only.

Example

```
For &I = 1 to &ROW.RecordCount
   &REC = &ROW.GetRecord(&I);
   /* do processing */
End-For;
```

RowNumber

Description

This property returns the row number (starting from 1) of the row within its rowset.

This property is read-only.

Example

The following returns the row number of the current effective-date row.

```
&NUMBER = GetRowset(SCROLL.MYRECORD).GetCurrEffRow().RowNumber;
```

Selected

Description

This property returns True if the row is part of a grid and has been selected either by the user or programmatically.

This property is meaningful in PIA only if the grid or scroll area attribute has been set to provide the selection control. In this case, the property reflects the state of the checkbox or radio button used for selection.

This property is read-write.

Example

The following example determines how many rows in a grid are selected.

```
Function Count_Child_Assets(&GRIDLEVEL1 As Rowset, &NUM_CHILDREN)
   REM *** How many child assets are selected?                  *;
   &GRIDLEVEL1 = GetRowset(SCROLL.PARENT_CHILD_VW);
   For &CHILDROW_COUNT = 1 To &GRIDLEVEL1.ActiveRowCount
      If &GRIDLEVEL1.GetRow(&CHILDROW_COUNT).Selected = True Then
         &NUM_CHILDREN = &NUM_CHILDREN + 1;
      End-If;
   End-For;
End-Function;
```

Style

Description

This property returns the name of the style class if one has been set for the row. You can also use this property to change the style of a row. This property takes a string value.

Note. This property *overrides* only the existing style. It doesn't *change* it. The next time the page is accessed the original style is used.

The PSSTYLE style sheet does *not* contain a default style class for a row. The Style property for all rows is initially NULL (that is, two quotation marks with no space between them ("")). The row assumes the style of the grid alternate row style. Fields in the row assume field styles if set.

Setting the property for a row overrides the grid alternate row style and any page field styles (field styles that were set in Application Designer). It does *not* override any field styles set (using the Field class Style property) for fields in the row.

Setting the Style property back to NULL (that is, two quotation marks with no space between them ("")) turns off the override.

Note. This property is *not* valid for Windows Client applications.

Example

```
&Scroll = GetLevel0().GetRow(1).GetRowset(Scroll.IC_PC_STYLE_2);

&row_class = &Scroll.GetRow(&row);

&row_class.Style = &style;
```

See Also

Chapter 18, "Field Class," Style, page 971

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Creating Field Definitions"

Visible

Description

If this property is True, the row is visible if displayed on the current page. This property can be set False to hide the row.

When Visible is set to False to hide a row, the row moves to the end of the rowset. This means that the row number of the row being hidden, and any subsequent rows, *changes* as a result of hiding a row. If the row is later made visible again, it is *not* moved back to its original position, but remains at the end of the rowset. It is just moved to the start of all the other hidden rows.

For example, if there are 4 rows in a rowset and row 2 is hidden (that is, Visible = False), that row now becomes row 4. In order to make that row visible again, row 4 must be set to Visible = True. Alternatively, you can use a row object reference: this remains valid even though the row number of the row may have been changed.

Note. If you use the Sort method on a rowset with hidden rows, the hidden rows aren't sorted. Only visible rows are sorted.

You cannot hide rows in the current context of the executing program. This means Visible cannot hide the row containing the executing program, or in a child rowset and be executed against its parent rowset. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Example

```
&ROW.Visible = False;
```


The following code uses the Visible property to determine if a row is visible or not before updating the value of the row.

```
&ROWSET = GetRowset(SCROLL.LD_SHP_INV_VW)

For &I = 1 To &ROWSET.ActiveRowCount
   If &ROWSET(&I).Visible Then
      ITEM_SELECTED.value = "N";
   End-If;
End-For;
```

The following code starts with the last row and works forward to the first. If any changes are made they don't impact the position of rows that still have to be processed.

```
For &I = &ACTIVE_STREAMS to 1 Step -1
   &Row = &STREAM_ROWSET(&I);
   If &Row.QS_STREAM8.STREAM_ROOT_ID.Value <> &STREAM_ROOT_ID Then
      &Row.Visible = False;
   End-if;
End-For;
```


Chapter 36

Rowset Class

This chapter provides an overview of Rowset class and discusses the following topics:

- Shortcut considerations.
- Rowset object declaration.
- Scope of a Rowset object.
- Rowset class built-in functions.
- Rowset class methods.
- Rowset class properties.

Understanding Rowset Class

A *rowset* object, instantiated from a Rowset class, is a collection of rows associated with buffer data. A component scroll is a rowset. You can also have a level zero rowset.

If a rowset object is instantiated using GetRowset (either the function or one of the methods) the rowset object that is instantiated is populated with data according to the context in which it was instantiated.

If a rowset object is instantiated using the CreateRowset function, the rowset object that's instantiated is a standalone rowset. Any records and field references created by this function are initialized to null values, that is, they do *not* contain any data. You can populate this rowset object using the CopyTo, Fill, or FillAppend methods.

Default processing isn't performed on a standalone rowset. In addition, a standalone rowset isn't tied to the Component Processor. When you fill it with data, no PeopleCode runs (for example, RowInsert, FieldDefault, and so on.) Delete and insert activity on these types of rowsets aren't automatically applied at save time. Use standalone rowsets for work records.

You might use a rowset object and the ActiveRowCount property to iterate over all the rows of the rowset, or to access a specific row (using the GetRow method) or to find the name of the primary record associated with the scroll (the DBName property).

The Rowset class is one of the data buffer access classes.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Shortcut Considerations

The default method for the Rowset class is GetRow. This means you can specify just a row number, and not use the GetRow method. For example, the following two lines of code are equivalent:

```
&MyRow = GetRowset()(5);
```

```
&MyRow = GetRowset().GetRow(5);
```

Data Type of a Rowset Object

Rowset objects are declared as type Rowset. For example,

```
Local Rowset &MYROWSET;
```

Scope of a Rowset Object

A Rowset can be instantiated from PeopleCode or using Java.

This object can be used anywhere you have PeopleCode, that is, in an Application Class, record field PeopleCode, and so on.

You can't pass a rowset object in as part of a Component Interface user-defined method. (Rowsets aren't common data structures outside of a PeopleSoft system.) However, within a user-defined method for a Component Interface you can use rowset objects.

In an Application Engine program, a rowset object is the equivalent of a record object that contains one row and a single record, that is, the State Record. PeopleSoft suggests using the Record object instead of a rowset object to obtain access to the State Record.

Messages have the same structure as rowsets, that is, hierarchical data structures composed of rows, records, and fields.

File objects can have the same structure as rowsets, that is, hierarchical data structures composed of rows, records, and fields.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

Rowset Class Built-In Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

Rowset Class Methods

In this section, we discuss the Rowset class methods. The methods are discussed in alphabetical order.

ClearDeletesChanges

Syntax

```
ClearDeletesChanges()
```

Description

Use the ClearDeletesChanges method to clear deleted rows and changed from a standalone rowset.

Note. This method works only with standalone rowsets, that is, rowsets created using the CreateRowset function.

This method differs from Flush in a number of ways:

- it does not remove *all* rows from the rowset, only deleted rows
- it only applies to standalone rowsets

This method first clears the deleted rows, that is, all rows that have been deleted using DeleteRow are removed from the rowset and their associated buffers are freed.

This method then clears changed rows. That means any changes done on a row (such as field values changed) or newly inserted rows are now propagated into their original state and the changed buffers, if any, are freed.

After executing this method on a standalone rowset, any row that was previously new or changed no longer has that state. The IsNew and IsChanged properties of a row return false.

This method does *not* do any database updates.

How would you use this method? Suppose you use a standalone rowset to track changes you need to make to some business process or object. After doing the appropriate database updates to reflect changes recorded in the rowset (that is, inserts or deletes or changes), you call this method to clean up the rowset in preparation for further processing. Without this method, newly inserted rows and changed rows preserve their IsNew and IsChanged status indefinitely, complicating program logic and potentially leading to duplicate inserts or deletes.

Parameters

None.

Returns

None.

Example

```
REM +--------------------------------------------------+;
REM | Function to Update the DB for a Standalone Rowset|;
REM +--------------------------------------------------+;
Function ProcessDatabaseUpdateforRowset(&rsIn As Rowset)

   For &i = 1 To &rsIn.RowCount

      &rwTMP = &rsIn.GetRow(&i);
      If &rwIn.IsDeleted And
            Not &rwIn.IsNew Then
         &rTMP = &rwIn.GetRecord(1);
         &rTMP.Delete();
      End-If;

      If &rwTMP.IsNew And
            &rwTMP.IsChanged And
            Not &rwTMP.IsDeleted Then

         &rTMP = &rwTMP.GetRecord(1);
         &rTMP.Insert()
      End-If;

      If Not &rwTMP.IsNew And
            &rwTMP.IsChanged And
            Not &rwTMP.IsDeleted Then

         &rTMP = &rwTMP.GetRecord(1);
         &rTMP.Update();
      End-If;

   End-For;

   REM +-----------------------------------------------+;
   REM | Now we need to reset the Rowset flags and     |;
   REM | remove deleted rows                           |;
   REM +-----------------------------------------------+;
   &rsIn.ClearDeletesChanges();

End-Function;
```

See Also

Chapter 36, "Rowset Class," Flush, page 2140; Chapter 36, "Rowset Class," FlushRow, page 2142; Chapter 35, "Row Class," IsChanged, page 2120 and Chapter 35, "Row Class," IsNew, page 2122

CopyTo

Syntax

CopyTo(*&DestRowset* [, *record_list*])

Where *record_list* is a list of record names in the form:

[**RECORD.***source_recname1*,**RECORD.***target_recname1*

```
[, RECORD.source_recname2, RECORD.target_recname2]]. . .
```

Description

The CopyTo method copies *from* the rowset executing the method *to* the specified destination rowset, copying *like-named* record fields and subscrolls at corresponding levels.

The CopyTo method uses the current data in the rowset. This might be different from the original data values if the rowset was retrieved from the database and values in it have been changed either by an end-user or a PeopleCode program

If pairs of source and destination record names are given, these are used to pair up the records and subscrolls *before* checking for like-named record fields. Then, after copying the named records pairs, this method copies *all* identically named records.

Note. This method does not work for Application Engine state records. If you don't specify *record_list,both* the record name *and* the field name have to match exactly for data to be copied from one record field to another. If you specify *record_list,* after the records have been paired up, the field names have to match before any data is copied.

If the rowset you are copying *from* has the field level EditError, as well as the MessageNumber and MessageSetNumber properties set, these values are copied to the rowset you are copying to. For example, suppose you had an application message that had errors. Using the GetSubContractInstance function, these errors would be copied into the message object. From there, you could instantiate a message rowset, copy the message rowset into a work rowset, and use the work rowset to populate Component buffers. (After the field properties are set, the Record, Row, and Rowset properties IsEditError also get set.)

Parameters

Parameter	***Description***
&DestRowset	Specify the rowset to be copied to. This rowset object must have already been instantiated.
SourceRecname	Specify a record to be copied from, in the rowset object being copied from.
DestRecname	Specify a record to be copied to, in the rowset object to be copied to.

Returns

None.

Example

If you set one rowset equal to another, you haven't made a copy of the rowset. Instead, you have two variables pointing to the *same* data.

To make a clone of an existing rowset, that is, to make two distinct copies, you can do the following:

```
&RS2 = CreateRowset(&RS);
&RS.CopyTo(&RS2);
```

The following example copies data from one rowset object to another. Because no like-named records exist between the two rowsets, the record names are specified. Only the like-named fields are copied from one rowset to the other:

```
Local Rowset &RS1, &RS2;
Local String &EMPLID;

&RS1 = CreateRowset(RECORD.PERSONAL_DATA);
&RS2 = CreateRowset(RECORD.PER_VENDOR_DATA);
&EMPLID = "8001";
&RS1.Fill("WHERE EMPLID =: 1", &EMPLID);
&RS1.CopyTo(&RS2, RECORD.PERSONAL_DATA, RECORD.PER_VENDOR_DATA);
```

The following example copies data from a message into the Component buffers, then calls the page (using TransferPage) to redraw the page. You could do this to fill a page with message data that is in error, so that an end-user can make corrections to the message data. &WRK_ROWSET0 is the level zero rowset and &WRK_ROWSET1 is where the data is copied to.

```
/* Get the Message */
&MSG = GetSubContractInstance(&PUBID, &PUBNODE, &CHNLNAME, &MSGNAME, &SUBNAME);

/* Get the Message Rowset */
&MSG_ROWSET = &MSG.GetRowset();

/* Get Level 0 */
&WRK_ROWSET0 = GetLevel0();

/* Create Work rowset */
&WRK_ROWSET1 = GetLevel0()(1).GetRowset(SCROLL.EN_REVISION_TMP);

/* Populate Work Rowset */
&MSG_ROWSET.CopyTo(&WRK_ROWSET1, RECORD.EN_REVISION, RECORD.EN_REVISION_TMP);

SetNextPage("EN_REVISION_MSG");

TransferPage();
```

See Also

Chapter 35, "Row Class," CopyTo, page 2113 and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," CreateRowset

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

DeleteRow

Syntax

`DeleteRow`*`(n)`*

Description

The DeleteRow method deletes the row in the rowset identified by the parameter.

If the program is being run from a component against Component buffer data, a RowDelete PeopleCode event also fires, followed by the events that normally follow a RowDelete, as if the user had manually pressed ALT+8 and ENTER.

This method initially marks the row as needing to be deleted. At save time the row is actually deleted from the database and cleared from the buffer. When the row is marked as deleted, it is ignored by other methods, such as GetCurrEffRow, Sort, and so on.

DeleteRow cannot be executed from the same rowset where the deletion takes place, or from a child rowset against a parent. Place your PeopleCode in a parent rowset and execute it against a child rowset.

When DeleteRow is used in a loop, you have to process rows from high to low to achieve the correct results, that is, you must delete from the bottom up rather than from the top down. This is necessary because the rows are renumbered after they are deleted (if you delete row one, row two becomes row one).

Note. If you use DeleteRow on a rowset created using the CreateRowset function, the row isn't automatically deleted in the database when the page is saved. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren't automatically applied at save time.

Parameters

Parameter	***Description***
n	An integer identifying a row within the rowset object. This must be >= 1 and <= the number of active rows in the rowset (see ActiveRowCount).

Returns

An optional Boolean value: True if row is deleted, False otherwise.

Example

In the following example DeleteRow is used in a For loop. The example checks a value in each row, then conditionally deletes the row. Note the syntax of the For loop, including the use of the -1 in the Step clause to loop from the highest to lowest values. This ensures that the renumbering of the rows do not affect the loop.

```
For &I = &RS2.ActiveRowCount To 1 Step -1
   If None(&CHECK_SEQ) Then
      &RS2.DeleteRow(&I);
   End-If;
End-For;
```


See Also

Chapter 36, "Rowset Class," Flush, page 2140; Chapter 36, "Rowset Class," FlushRow, page 2142; Chapter 36, "Rowset Class," InsertRow, page 2148 and Chapter 34, "Record Class," Insert, page 2090

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

Fill

Syntax

```
Fill([wherestring [, bindvalue] . . .])
```

Description

The Fill method flushes the rowset then reads records from the database into successive rows. The records are read from the database tables corresponding to the primary database record of the scroll into that record. The records are selected by the optional *wherestring* SQL clause, in which the optional *bindvalues* are substituted, using the usual bind placeholders *(:n)*.

In general, use this method only with rowsets that were created using the CreateRowset function.

Note. Because Flush always leaves one row in the scroll, there will be one row in the scroll even if you don't read any records.

The actual number of records read into the rowset is an optional return of this method.

Note. This method does not work with Application Engine state records. Also, you cannot use this method in dynamic views.

When this method executes, unlike the Select method, it does *not* cause any associated PeopleCode to run as part of reading data into the rowset.

Note. Fill reads only the primary database record. It does not read any related records, nor any subordinate rowset records.

For every record read with the Fill method, if the set language is not the base language and the record has related language records, the Fill method tries to read the related language record and does related language processing.

The Fill method uses a correlation ID of FILL for the table it reads. You must use the correlation ID if you want to refer to the rowset table name as part of the *wherestring.* You receive a runtime error if you use the table name as a column prefix instead of the correlation ID.

Sorting Considerations

Rows come unsorted from the database when using Fill. This is not a problem for SQL server, however, it can be a problem for DB2 UDB for OS/390 and z/OS and Oracle.

See Chapter 36, "Rowset Class," Sort, page 2160.

Parameters

Parameter	*Description*
wherestring	Specify a SQL WHERE clause to use for selecting records to fill the rowset. This can be a string or a SQL definition.
bindvalue	Specify optional bind variables to be used with the WHERE clause.

Returns

The number of records read into the rowset.

Example

The following example reads all of the QA_MYRECORD records into a rowset, and returns the number of rows read:

```
&RS = CreateRowset(RECORD.QA_MYRECORD);
&NUM_READ = &RS.Fill();
```

The following example reads all of the QA_MYRECORD records that have a MYRECORD field equal to the value of &UVAL into a rowset, and returns the number of rows read:

```
&NUM_READ = &RS.Fill("where MYRECORD = :1", &UVAL);
```

To re-use a WHERE clause for the *wherestring* you can use the SQL repository, and a SQL object.

```
&NUM_READ = &RS.Fill(SQL.MYWHERE, &UVAL);
```

The following example gets all the SET_CNTRL_REC rows related to the row on the page, then updates SETID with the value from the page. Fill is used with a rowset that was created from a message that was just created, that is, a rowset that was unpopulated.


```
If FieldChanged(SETID) Then
   &MSG = CreateMessage(OPERATION.SET_CNTRL_REC);
   &MSG_ROWSET = &MSG.GetRowset();
   &MSG_ROWSET.Fill("where SETCNTRLVALUE =:1 and REC_GROUP_ID =:2", SETCNTRLVALUE,⇒
 REC_GROUP_ID);

For &I = 1 To &MSG_ROWSET.ActiveRowCount
   &MSG_ROWSET.GetRow(&I).SET_CNTRL_REC.SETID.Value = SETID;
   &MSG_ROWSET.GetRow(&I).PSCAMA.AUDIT_ACTN.Value = "C";
End-For;

&MSG.Publish();

End-If;
```

When using the Fill method, the IsChanged property of each field in a part rowset is not set to true. Because the fields appear to be unchanged, this can create a problem for publication of data from Message rowsets. A technique to avoid this problem is to create a second rowset and use the CopyTo method to copy the changes to the Message rowset as shown in the following example:

```
&a = CreateMessage(Operation.MY_ASYNC);
&rs = &a.GetPartRowset(1);

&trs = CreateRowset(Record.PSPMAGENT);
&trs.Fill("where PM_AGENTID >= 12345");

&trs.CopyTo(&rs);
%IntBroker.Publish(&a);
```

The following example uses a correlation ID for the table in the Fill SELECT, to enable the use of correlated subqueries in the WHERE clause, such as the usual effective-date subquery:

```
&RSOWNER_NAME = CreateRowset(RECORD.PERSONAL_D00);
&RSOWNER_NAME.Fill("where SETID=:1 AND EMPLID=:2 AND %EffDtCheck(PERSONAL_⇒
D00,FILL,:3)", &SETID, &EMPLID, &EFFDT);
```

The Fill method implicitly uses Fill as an alias for the Rowset record. This is helpful for complex Fill *where* clauses with subqueries.

```
&oprclasscountries = CreateRowset(Record.SCRTY_TBL_GBL);

&oprclasscountries.Fill("WHERE FILL.OPRCLASS = :1 AND NOT EXISTS (SELECT 'X' FROM⇒
 PS_SCRTY_SEC_GBL GBL2 WHERE GBL2.OPRCLASS = FILL.OPRCLASS AND GBL2.COUNTRY =⇒
 FILL.COUNTRY AND GBL2.PNLNAME = :2)", &OPRCLASS, %Component);
```

In the following example, the necessary key field values are loaded into a rowset, then the following function is called, and the values are used as part of the Fill method.

```
Function FillRS2();

   Local SQL &MySql;
   Local string &MySqlString;
   Local Record &ElemDefnRec;
   Local string &ElemDefnRecName, &fldname;
   Local Rowset &CompRec2RS;

   /* Build the record object that is used for building the SQL and executing the⇒
 select */
   &ElemDefnRec = CreateRecord(@("Record." | &pkgRecName));

   /* Initialize the SQL string */
   &MySqlString = "%SelectByKey(:1 A)";

   /* Create a SQL to select a rows based on the key fields. */
   For &i = 1 To &ElemDefnRec.FieldCount
      If &ElemDefnRec.GetField(&i).IsKey Then
         &fldname = &ElemDefnRec.GetField(&i).Name;
         &ElemDefnRec.GetField(&i).Value = &CompRec2RS.GetRow(1).GetRecord(@⇒
("Record." | &pkgRecName)).GetField(@("Field." | &fldname)).Value;
      End-If;
   End-For;

   /* Create the SQL and execute the select */
   &MySql = CreateSQL(&MySqlString);
   &MySql.Execute(&ElemDefnRec);

   /* Copy each selected row into the rowset */
   While &MySql.Fetch(&ElemDefnRec)
      &ElemDefnRec.CopyFieldsTo(&CompRec2RS(&CompRec2RS.ActiveRowCount).GetRecord⇒
(1));
   End-While;

End-Function;
```

See Also

Chapter 36, "Rowset Class," CopyTo, page 2131; Chapter 36, "Rowset Class," Select, page 2154 and Chapter 36, "Rowset Class," FillAppend, page 2138

Chapter 40, "SQL Class," page 2239

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

PeopleTools 8.51 PeopleBook: Global Technology, "Using Related Language Tables"

FillAppend

Syntax

FillAppend([*wherestring* [, *bindvalue*] . . .])

Description

The FillAppend method reads records from the database into successive rows of the rowset starting *after* the last row that exists in the rowset. Unlike Fill, it doesn't first flush the rowset.

Note. FillAppend appends rows after the last *active* row in the rowset. If you have deleted rows in the rowset, they will still be the last rows.

When a rowset is selected into, any autoselected child rowsets are also read. The child rowsets are read using a where clause that filters the rows according to the where clause used for the parent rowset, using a subselect.

When a rowset is created using CreateRowset, it contains one empty row. If the rowset hasn't been filled with data, FillAppend fills that row also (so you don't have an empty row at the start of your rowset.)

The records are read from the database tables corresponding to the primary database record of the scroll into that record. The records are selected by the optional *wherestring* SQL clause, in which the optional *bindvalues* are substituted, using the usual bind placeholders *(:n).*

In general, use this method only with rowsets that were created using the CreateRowset function.

The actual number of records read into the rowset is an optional return of this method.

Note. This method does not work with Application Engine state records. Also, you cannot use this method in dynamic views.

When this method executes, unlike the Select method, it does *not* cause any associated PeopleCode to run as part of reading data into the rowset.

Note. FillAppend reads only the primary database record. It does not read any related records, nor any subordinate rowset records.

For every record read with the FillAppend method, if the set language is not the base language and the record has related language records, the FillAppend method tries to read the related language record and does related language processing.

The FillAppend method uses a correlation ID of FILLAPPEND for the table it reads. You must use the correlation ID if you want to refer to the rowset table name as part of the *wherestring.* You receive a runtime error if you use the table name as a column prefix instead of the correlation ID.

Parameters

Parameter	***Description***
wherestring	Specify a SQL WHERE clause to use for selecting records to fill the rowset. This can be a string or a SQL definition.
bindvalue	Specify optional bind variables to be used with the WHERE clause.

Returns

The number of records read into the rowset.

Example

The following example reads all of the QA_MYRECORD records that have a MYRECORD field equal to the value of &UVAL into a rowset, and returns the number of rows read:

```
&NUM_READ = &RS.FillAppend("where MYRECORD = :1", &UVAL);
```

To re-use a WHERE clause for the *wherestring* you can use the SQL repository, and a SQL object.

```
&NUM_READ = &RS.FillAppend(SQL.MYWHERE, &UVAL);
```

See Also

Chapter 36, "Rowset Class," CopyTo, page 2131; Chapter 36, "Rowset Class," Select, page 2154 and Chapter 36, "Rowset Class," Fill, page 2135

Chapter 40, "SQL Class," page 2239

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

PeopleTools 8.51 PeopleBook: Global Technology, "Using Related Language Tables"

Flush

Syntax

```
Flush()
```

Description

Use the Flush method to remove all rows from the rowset and free its associated buffer. Rows that are flushed are not deleted from the database. This function is often used to clear a work scroll before using the Select method.

Note. Flush always leaves one row in the scroll.

You cannot flush the current context of the executing program. This means Flush cannot be used for the rowset containing the executing program, or in any child rowset and executed against the parent rowset. Place your PeopleCode in a parent rowset and execute it against a child rowset.

For rowsets corresponding to component buffer data, the Flush method executes in turbo mode only—that is, default processing is performed, but only on the row being flushed. Additional information on turbo mode and non-turbo mode is available in the documentation with the InsertRow PeopleCode built-in function.

See *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," InsertRow.

Considerations When Initializing New Rows

Flush removes all rows and inserts a row.

If you want to initialize the row, that is, have the defaults and any RowInit PeopleCode run, you must do something to invoke the default value processing. This can be as simple as setting the value of another field on the same page that has a PeopleCode program associated with it.

If the rowset is created from message data, an Application Engine program, and so on, the rows are flushed but the row that is inserted is not initialized.

Considerations for User-Sorted Grids

If a grid has a user personalized sort defined for it and your PeopleCode program flushes that grid and then repopulates it (for instance, through a Fill, a Select, or a series of InsertRow calls), this could invalidate the user's personalized sort and the current row in the user's view unless your PeopleCode program makes arrangements to reapply these user personalizations.

After flushing and repopulating the rowset, your PeopleCode program should re-sort the rowset (that is, re-sort the grid). If the user has a personalized sort, then the user's personalized sort will be reapplied via the PeopleCode sort.

Also, after flushing and repopulating a grid, it will be important to manage which row is deemed the current row. This can be managed using methods such as IsUserSorted, GetFirstUserSortedRow, MapBufRowToUserSortRow, and then by setting the TopRowNumber property accordingly.

See Chapter 36, "Rowset Class," IsUserSorted, page 2149.

Parameters

None.

Returns

None.

Example

The following example checks for the value of the field CHECKLIST_CD. If something exists in this field, the second level rowset is flushed and new values are selected into it.

```
If All(CHECKLIST_CD) Then
   &RS1H.Flush();
   &RS1H.Select(RECORD.CHECKLIST_ITEM, "where Checklist_CD = :1 and EffDt = ⇒
(Select Max(EffDt) from PS_CHECKLIST_ITEM Where CheckList_CD = :2)", CHECKLIST_CD,⇒
 CHECKLIST_CD);
End-If;
```


See Also

Chapter 36, "Rowset Class," FlushRow, page 2142; Chapter 36, "Rowset Class," DeleteRow, page 2133 and Chapter 36, "Rowset Class," InsertRow, page 2148

FlushRow

Syntax

FlushRow*(n)*

Description

Use the FlushRow method to remove a specific row from a rowset and from the Component buffer. Rows that are flushed are not deleted from the database.

FlushRow is a specialized and infrequently used method. In most situations, you want to use DeleteRow to remove a row from the Component buffer and remove it from the database as well when the component is saved.

You cannot flush the current context of the executing program. This means FlushRow cannot be used for the rowset containing the executing program, or in any child rowset and executed against the parent rowset. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Parameters

Parameter	***Description***
n	An integer identifying a row within the rowset object. This must be >= 1 and <= the number of rows in the rowset (see property RowCount).

Returns

None.

Example

```
&ROWSET.FlushRow(&ROWNUMBER);
```

See Also

Chapter 36, "Rowset Class," DeleteRow, page 2133; Chapter 36, "Rowset Class," Flush, page 2140 and Chapter 36, "Rowset Class," InsertRow, page 2148

GetCurrEffRow

Syntax

GetCurrEffRow()

Description

GetCurrEffRow returns a row object for the row with the current effective date. This includes dates equal to the current date, but not dates in the future. If the primary record associated with the rowset is not effective-dated this method returns a null object.

Newly inserted rows are intentionally skipped when looking for the current effective-dated row. This is to find the current effective row from the data that already exists in the database. In other words, a row cannot be considered as a current effective row until it is saved. To find the current effective-dated row in data that has been newly inserted and not yet saved, use the GetNewEffRow method instead.

Parameters

None.

Returns

A row object for the row with the current effective date.

Example

```
&tmp =  &ROWSET.GetCurrEffRow().RowNumber;
/* note RowNumber is a property of the row class */
```

See Also

Chapter 36, "Rowset Class," DeleteEnabled, page 2163; Chapter 36, "Rowset Class," GetRow, page 2146; Chapter 35, "Row Class," GetNextEffRow, page 2114 and Chapter 36, "Rowset Class," GetNewEffRow, page 2145

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRow

GetFirstUserSortedRow

Syntax

GetFirstUserSortedRow*(GridName)*

Description

Use this method to get the first row from a sort view.

Parameters

Parameter	Description
GridName	Specify the grid name as a string in PAGE.RECORD format in which PAGE is the page name and RECORD is the name of the grid as defined in Application Designer. (In Application Designer, select Page Field Properties, General tab to view the value for the Page Field Name field. The Page Field Name field's value defaults to the primary record name for the level at which the grid occurs; however, the value can be changed by the application developer.)

Returns

A row object for the first row from the sort view.

If the grid has not been sorted, the output row object is equivalent to the first row from the rowset.

See Also

Chapter 36, "Rowset Class," IsUserSorted, page 2149

GetLastUserSortedRow

Syntax

`GetLastUserSortedRow``(`*`GridName`*` [, `*`visible`*`])`

Description

Use this method to get the last row from a sort view.

Parameters

Parameter	*Description*
GridName	Specify the grid name as a string in PAGE.RECORD format in which PAGE is the page name and RECORD is the name of the grid as defined in Application Designer. (In Application Designer, select Page Field Properties, General tab to view the value for the Page Field Name field. The Page Field Name field's value defaults to the primary record name for the level at which the grid occurs; however, the value can be changed by the application developer.)
visible	Specify, as an optional Boolean parameter, whether to return the absolute last row from the rowset, which could be hidden, or the last visible row from the sort view. If true, the last visible row is returned; otherwise, the absolute last row from the rowset is returned. The default value is false—that is, return the absolute last row.

Returns

A row object for the last row from the sort view *(visible* is true) or the absolute last row from the rowset (*visible* is false).

See Also

Chapter 36, "Rowset Class," IsUserSorted, page 2149

GetNewEffRow

Syntax

GetNewEffRow()

Description

GetNewEffRow returns a row object for the row with the effective date closest to the current date, including newly inserted rows. This means dates equal to the current date, but not future dates. If the primary record associated with the rowset is not effective-dated this method returns a null object.

To find the current effective-dated row from the data that already exists in the database, that is, only from saved rows and *not* a newly created row, use the GetCurEffRow method instead.

Parameters

None.

Returns

A row object for the row with the current effective date.

Example

```
Local Row &MyRow = &ROWSET.GetNewEffRow();
```

See Also

Chapter 36, "Rowset Class," GetRow, page 2146; Chapter 35, "Row Class," GetNextEffRow, page 2114 and Chapter 36, "Rowset Class," GetCurrEffRow, page 2143

Chapter 36, "Rowset Class," DeleteEnabled, page 2163

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRow

GetRow

Syntax

GetRow*(n)*

Description

GetRow returns a row object from a rowset. This is a default method for rowsets. This means that any rowset object, followed by a parameter list, acts as if GetRow is specified.

Parameters

Parameter	*Description*
n	An integer identifying a row within the rowset object. This must be >= 1 and <= the number of rows in the rowset (see property RowCount).

Returns

Returns a row object for the specified row of the rowset.

Example

In the following example, all of the lines of code do the same thing: they return the 5th row from the rowset (&ROWSET is a rowset object.)


```
&ROW = GetRowset().GetRow(5);
&ROW = GetRowset()(5);
&ROW = &ROWSET.GetRow(5);
&ROW = &ROWSET(5);
```

The following example loops through all the rows in a rowset:

```
Local rowset &LEVEL1;
Local row &ROW;

&LEVEL1 = GetRowset(SCROLL.EMPL_CHECKLIST);
For &I = 1 to &LEVEL1.ActiveRowCount
   &ROW = &LEVEL.GetRow(&I);
   /* do some processing */
End-For;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRow

HideAllRows

Syntax

HideAllRows()

Description

HideAllRows hides all rows of the rowset. Using this method is equivalent to a loop setting the visible property of each row of the rowset to False.

You cannot hide rows in the current context of the executing program. This means HideAllRows cannot hide the rowset containing the executing program, or in any child rowsets and be executed against the parent rowset. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Parameters

None.

Returns

None.

Example

The following example hides the second level scroll if a value exists in the NO_COMMENTS field in the first level of the scroll. The code is running from the first level of the scroll.


```
Local Rowset &RS1, &RS2;

&RS1 = GetRowset();
&RS2 = GetRowset(SCROLL.EMPL_CHKLST_ITM);

For &I = 1 to &RS1.ActiveRowCount
   If ALL(&RS1.GetRow(&I).EMPLOYEE_CHECKLIST.NO_COMMENTS) Then
      &RS2.HideAllRows();
   End-If;
/*other processing */
End-For;
```

See Also

Chapter 36, "Rowset Class," ShowAllRows, page 2159 and Chapter 35, "Row Class," Visible, page 2125

InsertRow

Syntax

InsertRow*(n)*

Description

InsertRow programmatically performs the ALT+7 and ENTER (RowInsert) function. InsertRow inserts a new row in the current rowset *after* the row specified by the parameter if the primary record for the rowset is not effective-dated or a standalone rowset. If the primary record for the rowset is effective-dated or a standalone rowset, the new row is inserted *before* the current row, and all values from the current row are copied into the new row, except for EFFDT, which is set to the current date.

In addition, InsertRow propagates the keys from the higher level (if any) into the inserted row.

If the program is being run from a component against Component buffer data, a RowInit PeopleCode event also fires, followed by the events that normally follow a RowInsert, as if the user had manually pressed ALT+7 and ENTER.

The InsertRow method can be executed against the same rowset where the insertion will take place.

For rowsets corresponding to component buffer data, the InsertRow method executes in turbo mode only—that is, default processing is performed, but only on the row being inserted. Additional information on turbo mode and non-turbo mode is available in the documentation with the InsertRow PeopleCode built-in function.

See *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," InsertRow.

InsertRow cannot be executed from the same rowset where the insertion will take place, or from a child rowset against a parent. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Note. If you use InsertRow on a rowset created using the CreateRowset function, the row isn't automatically inserted in the database when the page is saved. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren't automatically applied at save time.

Effective-Dated Row Considerations

When a row is inserted, if that row contains child scrolls, this method also inserts an empty row for any child scrolls. The effective-date field for this empty row is also empty. The current date is *not* used.

Parameters

Parameter	*Description*
n	An integer identifying a row within the rowset object. This must be >= 0 and <= the number of active rows in the rowset (see property ActiveRowCount). A value of 0 inserts in front of the first row.

Returns

An optional Boolean value: True if the row is inserted, False if the row is not inserted.

Example

In the following example, as the primary database record isn't effective-dated, the new row is inserted *after* the second row in the rowset.

```
&ROWSET.InsertRow(2);
```

See Also

Chapter 36, "Rowset Class," DeleteRow, page 2133 and Chapter 34, "Record Class," Delete, page 2085

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

IsUserSorted

Syntax

IsUserSorted*(GridName)*

Description

Use this method to determine whether a user sort has been performed on the specified grid.

A *user sort* is defined as a user either having clicked on a column heading on the grid or having applied a sort order on the grid's customization page. In the case of a user sort, a *sort view* is created based on the sort order defined by the user. Since this sort view is for display purposes only, the underlying buffer is not sorted and remains in the non-sorted order. Four additional methods (GetFirstUserSortedRow, GetLastUserSortedRow, MapBufRowToUserSortRow, and MapUserSortRowToBufRow) enable you to interrogate this sort view. These methods can be used to map the sort view to the buffer, to map the buffer to the sort view, to retrieve the first user-sorted row, and to retrieve the last user-sorted row.

Note. When a grid includes a hyperlink field that is user-sortable, the hyperlink label must be set in PeopleCode prior to allowing the user to perform the sort. Alternatively, the buffer can be sorted on the hyperlink field in PeopleCode upon entry into the page.

Parameters

Parameter	***Description***
GridName	Specify the grid name as a string in PAGE.RECORD format in which PAGE is the page name and RECORD is the name of the grid as defined in Application Designer. (In Application Designer, select Page Field Properties, General tab to view the value for the Page Field Name field. The Page Field Name field's value defaults to the primary record name for the level at which the grid occurs; however, the value can be changed by the application developer.)

Returns

A Boolean value: true if a user sort has been performed on the grid, false otherwise.

See Also

Chapter 36, "Rowset Class," GetFirstUserSortedRow, page 2143; Chapter 36, "Rowset Class," GetLastUserSortedRow, page 2144; Chapter 36, "Rowset Class," MapBufRowToUserSortRow, page 2150; Chapter 36, "Rowset Class," MapUserSortRowToBufRow, page 2151 and Chapter 36, "Rowset Class," Sort, page 2160

MapBufRowToUserSortRow

Syntax

```
MapBufRowToUserSortRow(GridName,number)
```

Description

Use this method to return the sort view row number that corresponds to the specified buffer row number.

Parameters

Parameter	*Description*
GridName	Specify the grid name as a string in PAGE.RECORD format in which PAGE is the page name and RECORD is the name of the grid as defined in Application Designer. (In Application Designer, select Page Field Properties, General tab to view the value for the Page Field Name field. The Page Field Name field's value defaults to the primary record name for the level at which the grid occurs; however, the value can be changed by the application developer.)
number	Specify the row number of the row in the buffer.

Returns

The row number, as a number, of the row in the sort view.

If the grid has not been sorted, the output row number is the same as the buffer row number.

Example

```
&szSortViewName = "QE_PIA_SORT_GRID1.L2RSSORTTEST";
Local Rowset &rsCurrent = GetLevel0()(1).GetRowset(Scroll.QE_PIA_SORTEST2)(1).Get⇒
Rowset(Scroll.QE_PIA_SORTEST3);

If &rsCurrent.IsUserSorted(&szSortViewName) Then
   WinMessage("QE_PIA_SORT_GRID1-Rowset1 User Sorted");
Else
   WinMessage("QE_PIA_SORT_GRID1-Rowset1 Not User Sorted");
End-If;

For &I = 1 To &rsCurrent.ActiveRowCount
   &index = &rsCurrent.MapBufRowToUserSortRow(&szSortViewName, &I);
   WinMessage("QE_PIA_SORT_GRID1-Rowset1, Buffer Row Index:  " | &I | ",  Sort Row⇒
 Index:  " | &index | ",  Description:  " | &rsCurrent(&I).QE_PIA_⇒
SORTEST3.DESCR30.Value);
End-For;
```

See Also

Chapter 36, "Rowset Class," IsUserSorted, page 2149

MapUserSortRowToBufRow

Syntax

MapUserSortRowToBufRow*(GridName, number)*

Description

Use this method to return the buffer row number that corresponds to the specified sort view row number.

Parameters

Parameter	Description
GridName	Specify the grid name as a string in PAGE.RECORD format in which PAGE is the page name and RECORD is the name of the grid as defined in Application Designer. (In Application Designer, select Page Field Properties, General tab to view the value for the Page Field Name field. The Page Field Name field's value defaults to the primary record name for the level at which the grid occurs; however, the value can be changed by the application developer.)
number	Specify the row number of the row in the sort view.

Returns

The row number, as a number, of the row in the buffer.

If the grid has not been sorted, the output row number is the same as the buffer row number.

Example

```
&szSortViewName = "QE_PIA_SORT_GRID1.L2RSSORTTEST";
Local Rowset &rsCurrent = GetLevel0()(1).GetRowset(Scroll.QE_PIA_SORTEST2)(2).Get⇒
Rowset(Scroll.QE_PIA_SORTEST3);

If &rsCurrent.IsUserSorted(&szSortViewName) Then
   WinMessage("QE_PIA_SORT_GRID1-Rowset2 User Sorted");
Else
   WinMessage("QE_PIA_SORT_GRID1-Rowset2 Not User Sorted");
End-If;

For &I = 1 To &rsCurrent.ActiveRowCount
   &index = &rsCurrent.MapUserSortRowToBufRow(&szSortViewName, &I);
   WinMessage("QE_PIA_SORT_GRID1-Rowset2, Description:  " | &rsCurrent(&index).QE_⇒
PIA_SORTEST3.DESCR30.Value);
End-For;
```

See Also

Chapter 36, "Rowset Class," IsUserSorted, page 2149

Refresh

Syntax

Refresh()

Description

Refresh reloads the rowset object from the database using the current page keys. This causes the page to be redrawn. The following code example refreshes the entire page:

```
GetLevel0().Refresh()
```

Important! Do not call GetLevel0().Refresh() from any context that is not at level 0. You are likely to leave invalid pointers to objects that have been cleared from the buffer, which can result in an application server crash.

If you only want a specific scroll to be redrawn, you can use the refresh method with that rowset. You don't have to use a level zero rowset with this method. This is particularly useful after using RemoteCall.

Note. If a scroll is marked as No Auto Select in Application Designer, Refresh will not work on it. The Refresh method clears the existing buffers and does a refresh from the database. You do *not* want to use this method if your rowset is based on message data. Instead, you can use the CopyTo Rowset method combined with TransferPage.

Parameters

None.

Returns

None.

Example

The following example refreshes the entire page.

```
&Mylevel0 = GetLevel0().Refresh();
```

The following example refreshes only the second level scroll of the page:

```
&RowSetLevel2.Refresh();
```


See Also

Chapter 36, "Rowset Class," CopyTo, page 2131

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," RemoteCall

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TransferPage

Select

Syntax

`Select``([`*`parmlist`*`],` **`RECORD.`***`selrecord`*` [, `*`wherestr, bindvars`*`])`

Where *paramlist* is a list of child rowsets, given in the following form:

`SCROLL.`*`scrollname1`*` [, `*`SCROLL.scrollname2`*`] . . .`

The first *scrollname* must be a child rowset of the rowset object executing the method, the second *scrollname* must be a child of the first child, and so on.

Description

Select, like the related method SelectNew, reads data from the database tables or views into either a row or rowset object.

Note. This method is valid only when used with rowsets that reference data from the component buffers. You cannot use this method with a message rowset, a rowset from an Application Engine state record, and so on. You can't use this method with rowsets created using the CreateRowset function (use the Fill method instead.)

Select automatically places child rowsets in the rowset object executing the method under the correct parent row. If it cannot match a child rowset to a parent row, an error occurs.

When a rowset is selected into, any autoselected child rowsets is also read. The child rowsets are read using a where clause that filters the rows according to the where clause used for the parent rowset, using a subselect.

If you don't specify any child rowsets in *paramlist,* Select selects from a SQL table or view specified by *selrecord* into the rowset object executing the method.

If you specify a child rowset in *paramlist,* Select selects from a SQL table or view specified by *selrecord* into the child rowset specified in *paramlist,* under the appropriate row of the rowset executing the method.

If the rowset executing the method is a level zero rowset, and you specify Select without specifying any child rowsets with *paramlist,* the method reads only a single row, because only one row is allowed at level zero.

The rowset executing the method *does not* have to be a level zero rowset, so the Select method can produce the functionality of both the ScrollSelect and RowScrollSelect functions.

Note. For developers familiar with previous releases of PeopleCode: If the rowset executing the method is a level zero rowset, and you specify a child rowset with *paramlist,* this method functions exactly like ScrollSelect. If the rowset executing the method is not a level zero rowset, and no child rowsets are specified with *paramlist*, the method acts like RowScrollSelect.

The record definition of the table or view being selected from is called the *select record*, and identified with **RECORD.***selrecord.* The select record can be the same as the primary database record associated with the rowset, or it can be a different record definition that has compatible fields. If you define a select record that differs from the primary database record for the rowset, you can restrict the number of fields that are loaded into the buffers on the client workstation by including only the fields that you actually need.

Select accepts a SQL string that can contain a WHERE clause restricting the number of rows selected into the object. The SQL string can also contain an ORDER BY clause to sort the rows.

Select and its related methods generate a SQL SELECT statement at runtime, based on the fields in the select record and the WHERE clause passed to them in the method parameters.

For rowsets corresponding to component buffer data, the Select method executes in turbo mode only**—**that is, default processing is performed, but only on the row being selected. Additional information on turbo mode and non-turbo mode is available in the documentation with the InsertRow PeopleCode built-in function.

See *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," InsertRow.

Parameters

Parameter	***Description***
paramlist	An optional parameter list, for specifying children rowsets of the rowset executing the method, as the rowset to be selected into. The first *scrollname* in *paramlist* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.
RECORD.*selrecord*	Specifies the select record. The *selrecord* record must be defined in Application Designer and be a build SQL table (using Build, Project) as a table or a view, unless *selrecord* is the same record as the primary database record associated with the rowset. *selrecord* can contain fewer fields than the primary record associated with the rowset, although it must contain any key fields to maintain dependencies with other records.
wherestr	Contains a WHERE clause to restrict the rows selected from *selrecord* and/or an ORDER BY clause to sort the rows. The WHERE clause can contain the PeopleSoft meta-SQL functions such as %DateIn() or %CurrentDateIn. It can also contain inline bind variables.
bindvars	A list of bind variables to be substituted in the WHERE clause. The same restrictions that exist for SQLExec exist for these variables. See SQLExec for further information.

Returns

The number of rows read (optional.) This counts only the lines read into the specified rowset. It does not include any additional rows read into any child rowsets of the rowset.

Example

The following example selects into the level zero rowset, only one row. Depending on the autoselect settings on any child scrolls, they may be reselected too.

```
&LEVEL0 = GetLevel0();
&LEVEL0.Select(RECORD.PERSONAL_DATA, "WHERE.  .  .");
```

The following example selects into the level one scroll specified. Depending on the autoselect settings on any child (level two) scrolls, they may also be selected.

```
&LEVEL0.Select(SCROLL.EMPL_CHECKLIST, RECORD.EMPL_CHECKLIST, "WHERE.  .  .");
```

The following example selects into the child scroll of the level one rowset. Each row fetched is placed under the appropriate row in &LEVEL1. Note that instead of hard-coding the WHERE clause, the SQL repository is used to access a SQL definition named SELECT_WHERE.

```
&LEVEL1 = &LEVEL0()(1).GetRowset(SCROLL.EMPL_CHECKLIST);
&LEVEL1.Select(SCROLL.EMPL_CHKLST_ITM, RECORD.EMPL_CHKLST_ITM, SQL.SELECT_WHERE);
```

The following example first flushes the hidden work scroll, then selects into it based on a field on the page.

```
&RS1H.Flush();
&RS1H.Select(RECORD.CHECKLIST_ITEM, "where Checklist_CD = :1 and EffDt = (Select⇒
 Max(EffDt) from PS_CHECKLIST_ITEM Where CheckList_CD = :2)", CHECKLIST_CD,⇒
 CHECKLIST_CD);
```

See Also

Chapter 36, "Rowset Class," SelectNew, page 2156 and Chapter 34, "Record Class," SelectByKey, page 2096

SelectNew

Syntax

SelectNew([*parmlist*], **RECORD**.*selrecord* [, *wherestr, bindvars*])

Where *paramlist* is a list of child rowsets, given in the following form:

SCROLL.*scrollname1* [, **SCROLL**.*scrollname2*] . . .

The first *scrollname* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.

Description

The SelectNew method is similar to the Select method, except that all rows read into the rowset are marked *new* so they are automatically inserted into the database at Save time. This capability can be used, for example, to insert new rows into the database by selecting data using a view of columns from other database tables.

Note. This method is valid only when used with rowsets that reference data from the Component buffers. You cannot use this method with a message rowset, a rowset from an Application Engine state record, and so on. You can't use this method with rowsets created using the CreateRowset function (use the Fill method instead.)

Parameters

Parameter	***Description***
paramlist	An optional parameter list, for specifying children rowsets of the rowset executing the method, as the rowset to be selected into. The first *scrollname* in *paramlist* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.
RECORD.*selrecord*	Specifies the select record. The *selrecord* record must be defined in Application Designer and be a build SQL table (using Build, Project) as a table or a view, unless *selrecord* is the same record as the primary database record associated with the rowset. *selrecord* can contain fewer fields than the primary record associated with the rowset, although it must contain any key fields to maintain dependencies with other records.
wherestr	Contains a WHERE clause to restrict the rows selected from *selrecord* and/or an ORDER BY clause to sort the rows. The WHERE clause can contain the PeopleSoft meta-SQL functions such as %DateIn() or %CurrentDateIn. It can also contain inline bind variables.
bindvars	A list of bind variables to be substituted in the WHERE clause. The same restrictions that exist for SQLExec exist for these variables. See SQLExec for further information.

Returns

The number of rows read (optional.) This counts only the lines read into the specified rowset. It does not include any additional rows read into any child rowsets of the rowset.

Example

The following example selects rows from DATA2 and reads them into a level one scroll. If the user saves the page, these rows are inserted into DATA1.

```
&LEVEL0.SelectNew(SCROLL.DATA1, RECORD.DATA2, "Where SETID = :1 and CUST_ID =:2",⇒
 CUSTOMER.SETID, CUSTOMER.CUST_ID);
```

If you use the same WHERE clause in more than one Select, you may want to use the SQL repository to store the SQL fragment. That way, if you ever want to change it, you will have to change it in only one place.

```
&LEVEL0.SelectNew(SCROLL.DATA1, RECORD.DATA2, SQL.Select_New);
```

See Also

Chapter 36, "Rowset Class," Select, page 2154 and Chapter 34, "Record Class," SelectByKey, page 2096

SetDefault

Syntax

SetDefault*(recname.fieldname)*

Description

The SetDefault method sets the value of the specified *recname.fieldname* to a null value in every row of the rowset. If this method is being run from a component against Component buffer data, the next time default processing is performed, this value is reinitialized to its default value: either a default specified in its record field definition or one set programmatically by PeopleCode located in a FieldDefault event. If neither of these defaults exist, the Component Processor leaves the field blank.

No default processing is performed against data that is not in the Component buffers.

Blank numbers correspond to zero on the database. Blank characters correspond to a space on the database. Blank dates and long characters correspond to NULL on the database. SetDefault gives each field data type its proper value.

Parameters

Parameter	***Description***
recname.fieldname	Specifies a field. The field must be in the specified record, on the rows of the rowset.

Returns

None.

Example

This example resets the PROVIDER to its null value. This field is reset to its default value when default processing is next performed:


```
If COVERAGE_ELECT = "W" Then
   &ROWSET.SetDefault(INS_NAME_TBL.PROVIDER);
End-if;
```

See Also

Chapter 18, "Field Class," SetDefault, page 946

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Default Processing

ShowAllRows

Syntax

`ShowAllRows``()`

Description

ShowAllRows "unhides" all rows of the rowset object executing the method. Using this method is equivalent to a loop setting the visible property of each row of the rowset to True.

ShowAllRows cannot be executed from the same rowset you want to display, or from a child rowset against a parent. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Parameters

None

Returns

None.

Example

```
&ROWSET.ShowAllRows();
```

See Also

Chapter 36, "Rowset Class," HideAllRows, page 2147 and Chapter 35, "Row Class," Visible, page 2125

Sort

Syntax

```
Sort([paramlist,] sort_fields)
```

Where *paramlist* is a list of child rowsets, given in the following form:

```
SCROLL.scrollname1 [, SCROLL.scrollname2] . . .
```

The first *scrollname* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.

And where *sort_fields* is a list of field specifiers in the form:

```
[recordname.]field_1, order_1 [, [recordname.]field_2, order_2]_
```

Description

Sort programmatically sorts the rows in a rowset. The rows can be sorted on one or more fields.

If you specify a child rowset in *paramlist*, Sort selects a set of child rowsets to be sorted. If you don't specify a child rowset in *paramlist,* the rowset executing the method is sorted.

The type of sort done by this function, that is, whether it is a linguistic or binary sort, is determined by the Sort Order Option on the PeopleTools Options page.

Note. The Sort method sorts only *visible* rows in a rowset. If the rowset you're sorting has hidden rows, the hidden rows are not sorted. Rows marked for deletion are also not sorted.

Parameters

Parameter	***Description***
paramlist	An optional parameter list, for specifying children rowsets of the rowset executing the method, as the rowset to be sorted. The first *scrollname* in *paramlist* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.
sort_fields	A list of field and order specifiers which act as sort keys. The rows in the rowset will be sorted by the first field in either ascending or descending order, then by the second field in either ascending or descending order, and so on.
[*recordname.*]*field_n*	Specifies a sort key field in the rowset. The *recordname* prefix is required if you've specified a field that is not on the current record.
order_n	A single-character string. "A" specifies ascending order; "D" specifies descending order.

Returns

None.

Example

The first example repopulates a rowset in a page programmatically by first flushing its contents, selecting new contents using Select, then sorting the rows in ascending order by EXPORT_OBJECT_NAME:

```
Function populate_rowset;

   &RS1 = GetLevel0()(1).GetRowset(SCROLL.EXPORT_OBJECT);
   &RS1.Flush();
   &RS1.Select(RECORD.EXPORT_OBJECT, "where export_type =:EXPORT_TYPE_VW.EXPORT_⇒
TYPE");
   &RS1.Sort(EXPORT_OBJECT_NAME, "A");

End-Function;
```

See Also

Chapter 36, "Rowset Class," IsUserSorted, page 2149 and Chapter 36, "Rowset Class," Select, page 2154

Chapter 35, "Row Class," GetRowset, page 2117

Rowset Class Properties

In this section, we discuss the Rowset class properties. The properties are discussed in alphabetical order.

ActiveRowCount

Description

This property returns the number of active (non-deleted) rows in the rowset.

This property returns a value of 1 for an empty scroll (Flush always leaves an empty row.) You can use the IsChanged or IsNew properties to verify if the row is new.

This property is read-only.

Example

```
&tmp = &ROWSET.ActiveRowCount;
```


ChangeOnInit

Description

Normally, if a field value is changed, whether through PeopleCode or by an end user, the IsChanged property for the row is set to True. The exception to this is when a change is done in the FieldDefault or FieldFormula event, that is, if a value is set in one of these events, the row is not marked as changed.

At save time, all newly inserted *and* changed rows are written to the database. All newly inserted but *not* changed rows are *not* written to the database.

Use this property to specify whether a change made to a new row from RowInit or RowInsert PeopleCode is marked as changed. This property takes a Boolean value. The default value is True.

Setting this property to False causes changes to fields for the rowset done in RowInit and RowInsert PeopleCode to *not* mark a new row as IsChanged.

This property is read-write.

Runtime Considerations

You must set this property *before*anything is changed for a row, otherwise the row is marked as changed.

This property propagates to child rowsets. That is, if it's set for a parent rowset before doing an insert row, it's automatically set on any and all child rowsets.

In addition, parent rows are marked as changed if this property is set only for the child rowset. That is, if you've changed a level three child of a level two row, the level two and level one rows are marked as changed to maintain data integrity.

DataAreaCollapsed

Description

This property specifies whether the view of a data area is collapsed or expanded.

Note. You must set the Collapsible Data Area field on the properties for the level-based control for this property to have any effect.

This property changes to reflect the current state of the data area, according to whether the user has collapsed or expanded it. Changing the value collapses or expands the data area, but it does *not* prevent the user from collapsing (or expanding) it themselves.

Note. Because the user can change the value of this property, whatever value is set in PeopleCode isn't guaranteed to be still set the next time it is checked, because the user may have collapsed or expanded the data area in the meantime.

This property overwrites the value of the Default Initial View to Expanded field set in Application Designer. For example, if Default Initial View to Expanded is selected in Application Designer, then the value for the DataAreaCollapsed property is set to True, the control initially displays collapsed.

This property takes a Boolean value: True, initially display the data area collapsed, False, initially display the data area expanded.

This property is read-write.

Note. To collapse just a group box, use the DataAreaCollapsed field method.

See Also

Chapter 18, "Field Class," DataAreaCollapsed, page 947

Example

The following example checks the number of rows in the level one rowset. If the number of rows returned is larger than 100, the data area is initially displayed collapsed.

```
If &Level1.RowCount > 100 Then
   &Level1.DataAreaCollapsed = True;
Else
   &Level1.DataAreaCollapsed = False;
End-If;
```

DBRecordName

Description

This property returns the name of the primary database record as a string for the rowset.

This property is read-only.

Example

```
&DBNAME = &ROWSET.DBRecordName;
```

DeleteEnabled

Description

This property determines whether a rowset allows rows to be deleted (the equivalent of the user pressing ALT+8 and ENTER). This property takes a Boolean value.

Note. This property controls only whether an end-user can delete a row. Rows can still be deleted using PeopleCode.

The initial value of this property depends on how the scroll was created at design time. If the No Row Delete setting is selected on the Use tab of the scroll properties dialog box, DeleteEnabled will be False: otherwise it will be True.

Note. If No Row Delete is selected in Application Designer, setting DeleteEnabled to True will *not* override this value. To control whether a rowset allows deletions at runtime, you should *not* select No Row Delete at design time.

For consistency, PeopleSoft recommends that either all rowsets at a level should disable deletions, or they should all allow deletions.

For rowsets created with non-Component Processor data (such as message rowsets, Application Engine rowsets, and so on) this property has no effect.

Note. Don't use this property with rowsets that are created using CreateRowset. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren't automatically applied at save time.

This property is read-write.

See Also

Chapter 36, "Rowset Class," DeleteEnabled, page 2163 and Chapter 36, "Rowset Class," InsertEnabled, page 2165

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Using Scroll Areas, Scroll Bars, and Grids," Using Scroll Areas and Scroll Bars

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

EffDt

Description

This property references the effective date of the primary record associated with the rowset. If the primary record associated with the rowset is not effective-dated, this property has a null value. To find the primary record associated with a rowset object, you can use the DBRecordName property.

Note. This property isn't valid with rowsets created using the CreateRowset function.

This property is read-only.

Example

```
&tmp = &ROWSET.EffDt;
```


See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

EffSeq

Description

This property references the effective-sequence number of the primary record associated with the rowset. If the primary record associated with the rowset does not have an effective-sequence number, this property has the value 0. To find the primary record associated with a rowset object, you can use the DBRecordName property.

Note. This property isn't valid with rowsets created using the CreateRowset function.

This property is read-only.

Example

```
&tmp = &ROWSET.EffSeq;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

InsertEnabled

Description

This property determines whether a rowset allows rows to be inserted (the equivalent of the user pressing ALT+7 and ENTER). This property takes a Boolean value.

Note. This property controls only whether an end-user can insert a row. Rows can still be inserted using PeopleCode.

The initial value of this property depends on a value set at design time. If the No Row Insert setting is selected on the Use tab of the scroll properties dialog box, InsertEnabled is False: otherwise it is True.

Note. If No Row Insert is selected in Application Designer, setting InsertEnabled to True does *not* override this value. To control whether a rowset allows inserts at runtime, you should *not* select No Row Insert at design time.

For consistency, PeopleSoft recommends that either all rowsets at a level should disable inserts, or they should all allow inserts.

For rowsets created with non-Component Processor data (such as message rowsets, Application Engine rowsets, and so on) this property has no effect.

Note. Don't use this property with rowsets created using CreateRowset. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren't automatically applied at save time.

This property is read-write.

See Also

Chapter 36, "Rowset Class," DeleteEnabled, page 2163

PeopleTools 8.51 PeopleBook: PeopleSoft Application Designer Developer's Guide, "Using Scroll Areas, Scroll Bars, and Grids," Using Scroll Areas and Scroll Bars

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

IsEditError

Description

This property is True if an error has been found on any field in any record in any row or child rowset of the current rowset after executing the ExecuteEdits method on either a message object or a record object. This property can be used with the Field Class properties EditError (to find the field that's in error), MessageSetNumber and MessageNumber to find the error message set number and error message number.

This property is read-only.

Example

The following is an example showing how IsEditError, along with ExecuteEdits could be used:

```
&REC.ExecuteEdits();
If &ROWSET.IsEditError Then
   For &I = 1 to &ROWSET.ActiveRowCount
      &ROW = &ROWSET(&I);
         For &J to &ROW.RecordCount
            &REC = &ROW.GetRecord(&J);
               For &K = 1 to &REC.FieldCount
                  If &REC.GetField(&K).EditError Then
                     LOG_ERROR();
                        /* application specific call */
                  End-If;
            End-For;
         End-For;
   End-For;
End-If;
```


See Also

Chapter 34, "Record Class," ExecuteEdits, page 2086; Chapter 34, "Record Class," IsEditError, page 2106; Chapter 18, "Field Class," EditError, page 951; Chapter 18, "Field Class," MessageNumber, page 962 and Chapter 18, "Field Class," MessageSetNumber, page 963

Level

Description

This property returns the level, that is, the nesting depth, of the rowset object. The top-level rowset has a level number of 0.

This property is read-only.

Example

```
&tmp = &ROWSET.Level;
```

Name

Description

This property refers to the name of the rowset. This property returns different values, based on the type of rowset.

Type of Rowset	*Returns*
Rowset created using GetLevel0	returns an empty string
Component buffer rowset	returns primary record name
Message rowset	returns primary record name
Component Interface rowset	returns primary record name
Application Engine rowset	always returns AESTATE

This property is read-only.

Example

```
&tmp = &ROWSET.Name;
```

ParentRow

Description

This property returns a row object containing a reference to the parent row, that is, the row containing the rowset. If this is a top-level rowset (level zero), the ParentRow property has a null value.

This property is read-only.

Example

```
&tmp = &ROWSET.ParentRow.RowNumber;
/* note that RowNumber is a property of the row class */
```

ParentRowset

Description

This property returns a rowset object containing a reference to the parent rowset, that is, the rowset containing the rowset. If this is a top-level rowset (level zero), the ParentRowset property has a null value.

This property is read-only.

Example

```
&tmp = &ROWSET.ParentRowset.Level;
/* note Level is another property of the rowset class */
```

RowCount

Description

This property returns the total number of rows in the rowset. It includes deleted rows. (The ActiveRowCount property doesn't include deleted rows.)

This property is read-only.

Example

```
&tmp = &ROWSET.RowCount;
```

SetComponentChanged

Description

This property determines whether any changes to the rowset using PeopleCode marks the component as changed and the data written to the database if the rowset is not a based on a derived/work record.

This property takes a Boolean value: true, changes to the rowset mark the component as changed, false, the component is treated as if unchanged. The default is true.

If you set this property to false, this means that no row insert, row delete, field change on the rowset using PeopleCode would cause the system to treat the component data as changed.

The SetComponentChanged rowset class property overrides the SetComponentChanged field class property. If a field has SetComponentChanged set to true, but its associated rowset has SetComponentChanged set to false, any change to the field does not mark the component as changed.

This property is read-write.

See Also

Chapter 18, "Field Class," SetComponentChanged, page 967

TopRowNumber

Description

This property returns the row that is being displayed at the top of the scroll (if any) for the rowset.

Generally, this property is used to return the top row number of a scroll. However, sometimes you want to reposition the scroll. For example, if you use SetCursorPos to move the focus to a given field, there is no guarantee that the row containing the field is at the top of the scroll.

This property is read-write.

Chapter 37

RowsetCache Class

This chapter provides an overview of the RowsetCache class and discusses:

- Using the RowsetCache class
- Data type of a RowsetCache object
- Scope of a RowsetCache object
- RowsetCache object reference

Understanding a Rowset Cache

Many PeopleSoft applications use a metadata based design, where application PeopleCode reads metadata from a database record or some other source, then presents the user with an interface that is modified based on the metadata.

PeopleTools stores application data in a database cache to increase system performance. The RowsetCache class enables you to access this memory structure, created at runtime, and shared by all users.

Data that would qualify as metadata and would be appropriate for the RowsetCache class would be data that was commonly used yet fairly static, for example, country or currency tables.

The RowsetCache class does not provide a mechanism for synchronizing the data between database tables and the cache. If the underlying data changes, you must repopulate the rowset cache.

Note. Non-base language users may see different performance due to language table considerations.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Improving Your PeopleCode," Setting MaxCacheMemory

PeopleTools 8.51 PeopleBook: System and Server Administration, "Setting Application Server Domain Parameters," Cache Settings

Using the RowsetCache Class

The first time a user accesses a database, the rowset cache for that database is automatically created and populated. After that first user, all users and applications can take advantage of the rowset cache.

If the underlying data for a rowset cache changes, at the very least your application must delete the existing rowset cache. Your application can also delete, then recreate and repopulate the rowset cache if it changes the underlying data. For example:

```
Local Rowset &RS;
Local RowsetCache &CRS;
Local Boolean &RSLT = False;

Repeat
   &I = &I + 1;
   If &RS(&I).IsChanged Then
      &RSLT = True;
   End-If;
Until &RSLT or
   &RS.RowCount = &I;

If &RSLT Then
   &CRS = GetRowsetCache(&RS);
   &CRS.Delete();
   &CRS = CreateRowsetCache(&RS);
   &CRS.Save();
End-if;
```

PeopleSoft recommends that any standard operations used by your application, such as deleting and repopulating the rowset cache, be encapsulated in an application class.

See Chapter 6, "Application Classes," page 189.

The RowsetCache contains a rowset, and serializes the contained rowset to binary form for speed. As with most PeopleTools objects, RowsetCache objects are cached to memory and file, but they are also cached to the data base.

RowsetCache objects are intended to be local to a data base : they should not be transferred to another data base.

Error Handling

Every time you use the GetRowsetCache function, you should verify that the function executed successfully by testing for a null object. For example:

```
Local RowsetCache &RSC;

&RSC = GetRowsetCache(Rowset.MyRowset);

If (&RSC <> NULL) Then
   /* do processing */
Else
   /* call to populate rowset cache */
End-if;
```


Data Type of a RowsetCache Object

Cached rowset objects are declared as type RowsetCache. For example,

```
Local RowsetCache &Cache;
```

Scope of a RowsetCache Object

A RowsetCache object can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, Component Interface PeopleCode, and so on.

Note. PeopleSoft recommends that you do not specify RowsetCache objects as Global, due to their potential size.

RowsetCache Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowsetCache

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowsetCache

RowsetCache Class Methods

In this section, we discuss the RowsetCache class methods. These methods are discussed in alphabetical order.

Delete

Syntax

Delete()

Description

Use the Delete method to delete the RowsetCache from memory. The RowsetCache must have already been created, either using the CreateRowsetCache and saving the RowsetCache, or by a user using the application.

If this method is called as part of a larger transaction, such as in the SavePreChange or SavePostChange event, the commit is tied to the commit of the outer transaction. If an error occurs in this method, the outer transaction will be rolled back.

If this method is invoked in a non-transactional PeopleCode event (such as RowInit) the commit is controlled by the regular Component Processor flow.

Parameters

None.

Returns

A Boolean value: True if the delete was successful, False if the specified RowsetCache wasn't found.

This method terminates with an error message if there was another problem.

Example

```
Local RowsetCache &CRS;
Local Boolean &RSLT;

&CRS = GetRowsetCache(Rowset.MyRowset);

/* do processing*/

If (&CRS <> NULL) Then
   &RSLT = &CRS.Delete();
End-If;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowsetCache and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetRowsetCache

Get

Syntax

Get()

Description

Use the Get method to convert a RowsetCache object into a rowset object. After using this method, you can manipulate the data in the rowset like you would any other rowset, using the regular rowset methods and properties.

Parameters

None.

Returns

A rowset object populated with the cached rowset data.

Example

```
Local Rowset &RS;
Local RowsetCache &CRS;

&CRS = GetRowsetCache(Rowset.MyRowset);

If (&CRS <> NULL) Then
   &RS = &CRS.Get();
Else
   /* do error processing */
End-if;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowsetCache

Chapter 36, "Rowset Class," page 2127

Save

Syntax

Save([[**Rowset.**]*name*] [, *Description*] [, *Lang*])

Description

Use the Save method to save a new or existing rowset to the cache. If a rowset of the specified name already exists, it will be replaced. Otherwise, the rowset will be added.

If this method is called as part of a larger transaction, such as in the SavePreChange or SavePostChange event, The commit is tied to the commit of the outer transaction. If an error occurs in this method, the outer transaction will be rolled back.

If this method is invoked in a non-transactional PeopleCode event (such as RowInit) the commit is controlled by the regular Component Processor flow.

Parameters

Parameter	*Description*
Rowset.*name*	Specify the name of the rowset to be saved to the cache. If you just specify *name*, you must enclose the name in quotation marks.
Description	Specify a text string describing the rowset.
Lang	Specify the language to use when saving the rowset. See the values below.

The values for the *Lang* parameter are:

Value	*Description*
%RowsetCache_SignonLang	Attempt to save the rowset to the cache using the sign-in language. However, if the base language rowset doesn't already exist in the cache, then the save to the sign-in language fails. This is because the system requires that base language data exist before related language data can be saved. If the *Lang* parameter is omitted, this is the default behavior.
%RowsetCache_BaseLang	Save the rowset to the cache using the base language.
%RowsetCache_SignonOrBaseLang	Attempt to save the rowset to the cache using the sign-in language. If that fails, then save the rowset using the base language.

Returns

A Boolean value: True if the save was successful, False otherwise.

Example

The following code cashes &RS as a rowset definition.

```
Local RowsetCache &CRS;
Local Rowset &RS;

&CRS = GetRowsetCache(Rowset.MyRowset);

If (&CRS <> Null) Then
   &RS = &CRS.Get();
. . . /* do processing */
Else
   /* do error processing */
End-if;

&CRS.Save();
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowsetCache

RowsetCache Class Properties

In this section, we discuss the RowsetCache class properties These properties are discussed in alphabetical order.

Content

Description

Use this property to specify the contents of a RowsetCache object.

This property is read-write.

Example

The following code example verifies if the rowset cache is populated. If it isn't, the Else statement populates the rowset cache.

```
&Cache2 = GetRowsetCache("CACHE1");
If &Cache2.Get() <> Null Then
   WinMessage("Cache found");
Else
   &RS = CreateRowset(Record.PSLANGUAGES);
   &NUM_READ = &RS.Fill();
   &Cache2.Content = &RS;
   &Cache2.Description = "test " | %Language_Data;
   &RSLT = &Cache2.Save();
End-If;
```

Description

Description

This property can be used to set the description of the rowset cache as a string.

This property is read-write.

Chapter 38

Session Class

This chapter provides an overview of Session class and discusses the following topics:

- Error handling.
- Regional settings.
- Trace settings.
- Session object declaration.
- Scope of a Session object.
- Session class built-in function.
- Session class reference section.

Understanding Session Class

PeopleTools provides a family of APIs for external access into the PeopleSoft system. The Session class is the root of the APIs. It controls access to the PeopleSoft system, controls the environment, and enables you to do error handling for all APIs from a central location.

The PeopleSoft APIs are intended to be used, and are supported, in the following environments.

- PeopleSoft Windows client with either an existing two-tier connection or an existing connection to an application server.
- Application Engine batch program.
- Non-PeopleSoft program connecting to an application server.

The PeopleSoft APIs are exposed to the following language environments:

- PeopleCode
- OLE/COM
- C/C++
- Java

The APIs that are accessible using a session object are:

- Component Interface Classes

- PortalRegistry Classes
- Query Classes
- Search Class
- Tree Classes

Note. Tree Classes are accessible only through PeopleCode.

A session object controls the following:

- Security and access to the PeopleSoft system.
- Errors and error handling .
- Regional settings (that is, internationalization) for language, dates, times, and numbers.
- Tracing.

See Also

Chapter 12, "Component Interface Classes," page 599

Chapter 30, "Portal Registry Classes," page 1587

Chapter 33, "Query Classes," page 1885

Chapter 38, "Session Class," page 2179

Chapter 43, "Tree Classes," page 2291

Security and Access to the PeopleSoft System

A session object controls security and access to the PeopleSoft system. You must "sign-on" to the PeopleSoft session with a valid user ID and password when you use the Connect method. In addition, there's an external authentication parameter that you can use with the connect.

Note. Additional authentication will be available after this release.

Error Handling

The session object handles error processing for all the APIs, such as Component Interfaces, the Query API, and so on. From the session object, you can check if there have been any errors.

Note. The exception to this is Subscription PeopleCode. All errors for Subscription PeopleCode get logged to the application message error table.

Error messages include system error messages or messages from the message catalog (a common set of error messages used by all PeopleSoft applications.) For a Component Interface, the session object level errors also contain any text errors that may have occurred from running ExecuteEdits.

On the session object, you can use the following properties to initially check for errors:

- ErrorPending indicates whether there are API errors
- WarningPending indicates whether there are API warnings

All errors are contained in the PSMessages collection. (The PSMessages property on a Session object returns this collection.)

Each item in this collection is a PSMessage object. A PSMessage object contains information about the specific error that has occurred, such as the explain text for the error, the message set number, and so on. (The type of information depends on the type of error.)

If the error was caused by a Component Interface, a contextual string is attached to the end of the Text property of the PSMessage object. This contextual string contains the exact location of the error, that is, which field in which data collection, on which row, caused the error. This string is also accessible (by itself) using the Source property.

When errors are loaded into the PSMessages collection depends on the type of error. System errors (that is, errors in the connection to the PeopleSoft Session) are logged as they occur. When an API error is logged depends on the API, as some APIs can be run in either interactive mode (meaning errors are logged as they happen) or in non-interactive mode (so errors are logged only when a particular method is run.)

If you are using Visual Basic or another COM environment, the PeopleSoft API system raises a COM exception the first time ErrorPending changes from False to True. It will *not* raise another exception until the PSMessages collection is cleared, which sets ErrorPending back to False.

Note. If you've called an API from an Application Engine program, all errors are also logged in the Application Engine error log tables.

See Also

Chapter 38, "Session Class," Accessing a PSMessages Collection, page 2191

Displaying Error Messages

Use the following PeopleCode to display messages from the PSMessage collection:

```
Local ApiObject &Session, &PSMessages;
   &Session = %Session;
   &PSMessages = &Session.PSmessages;
   If (&Session <> Null ) Then
       If &PSMessages.Count > 0 Then
               For &i = 1 To &PSMessages.Count
                  &MsgSetNbr = &PSMessages.Item(&i).MessageSetNumber;
                  &MsgNbr = &PSMessages.Item(&i).MessageNumber;
                  MessageBox( 0 , " ", &MsgSetNbr, &MsgNbr, "Message not found");
               End-For;
         &PSMessages.DeleteAll();
            End-If;
End-If;
```


Regional Settings

Regional settings enable the user to set the display of numbers, dates, times, and currencies to comply with usage in a specific locale. The session object exposes some of these values to the API through the RegionalSettings property. This property returns a regional settings object that has properties you can use to change these settings.

See Also

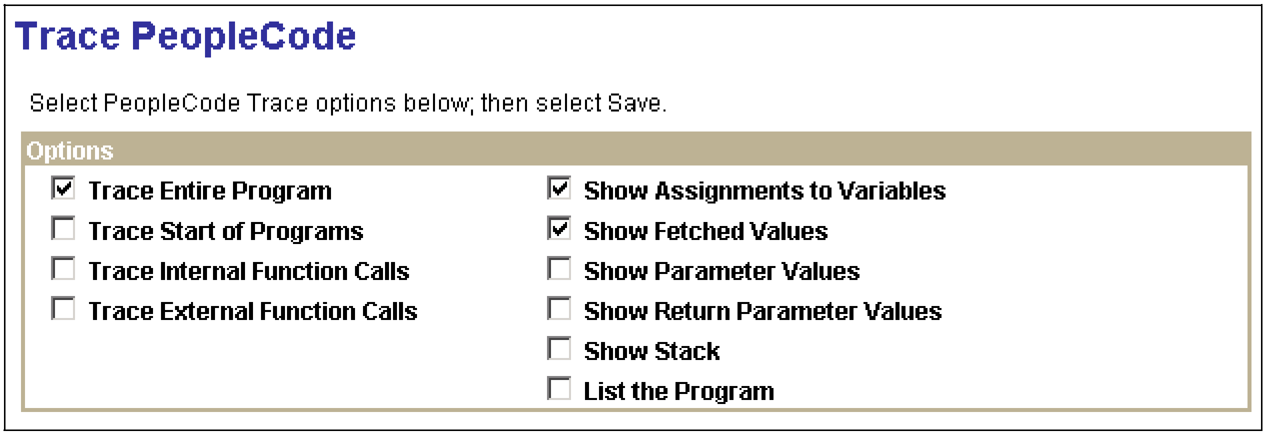
Chapter 38, "Session Class," Regional Settings Class Properties, page 2198

Trace Settings

The PeopleSoft debug pages enables you to control the environment in which your PeopleTools session is running. These pages include:

- Trace PeopleCode
- Trace SQL
- Trace Page

These pages enable you to select the SQL trace options and the PeopleCode trace options that you want. The TraceSettings object (assessable from a session object) lets you control many of the same options during your session.



Trace PeopleCode page

Not every option on the debug pages is represented with a trace settings property.

See Also

PeopleTools 8.51 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Using Debug Utilities

Session Object Declaration

All session objects are declared as type ApiObject. For example,

```
Local ApiObject &MYSESSION;
Global ApiObject &PSMessage;
```

The following session objects can be declared as Global or Component:

- Session
- PSMessages Collection
- PSMessage

All other session objects *must* be declared as Local.

In this section, we discuss the following topics:

- State considerations.
- Considerations for declaring conditions.

State Considerations

In the PeopleSoft Pure Internet Architecture, all processing occurs on the server. If you do something in your PeopleCode program that causes a trip back to the user before the end of your PeopleCode program, the PeopleCode program *cannot*regain its state, that is, it can't reset local variables, and so on. This applies to all APIs that use the Session object.

If you must go back to the user before the end of your PeopleCode program, you must set all your objects to NULL before you go, or else you receive a runtime error.

For example, the following code causes an error:

```
&Session = %Session;
Rem &Session = Null;
WinMessage("Hello");
```

The following example does *not* cause an error:

```
&Session = %Session;
&Session = Null;
WinMessage("Hello");
```


Considerations for Declaring Collections

You *must* declare all collections for C/C++ and COM as objects, or else you receive errors at runtime.

For example if you're using RegionalSettings, include the following your declarations for a VB program:

```
Dim oRegionalSettings As Object
```

Then, use the following code to populate this variable:

```
Set oRegionalSettings = oCISession.RegionalSettings
oRegionalSettings.LanguageCd = "GER"
```

Scope of a Session Object

A Session can be instantiated from PeopleCode, from a Visual Basic program, from C++, COM, and so on.

See the example section in Connect for more details.

This object can be used anywhere you have PeopleCode, that is, in Application Engine PeopleCode, record field PeopleCode, and so on.

Session Class Built-in Function

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "System Variables," %Session

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetSession

Session Class Methods

In this section, we discuss the Session class methods. The methods are discussed in alphabetical order.

Connect

Syntax

```
(version, {"EXISTING" | //PSoftApplicationServer:JoltPort}, UserID,Password,
ExtAuth)
```

Description

The Connect method connects a session object to a PeopleSoft application server.

Note. This function remains for backward compatibility only. Use the %Session system variable to return a reference of a session object instead.

If you already have a PeopleSoft session running (you are already connected to a PeopleSoft application server and are running a component, and so on) you must specify EXISTING, and not the *//PSoftApplicationServer:JoltPort.* Typically, use the parameter value of EXISTING from PeopleCode, not from other language environments.

If you are using an existing connection to the application server, you *cannot*specify a different user ID or password. If you don't specify these values as NULL string, you must specify the exact same user ID (and password) as the one that originally started the session.

Parameters

Parameter	Description
version	Specify the API version that the client is expecting. Future releases will use this parameter. For now, you must use a 1.
EXISTING | *//PSoftApplicationServer:JoltPort*	Specify EXISTING if you're currently connected to an application server. Otherwise, specify the named application server machine and the IP port to connect to on an application server machine.
UserID	Specify the PeopleSoft user ID to use for the connection. This must be a valid user ID. If you are using an existing connection, you can specify a NULL string (that is, two quotation marks with no blank space between them ("")) for this parameter.
Password	Specify the password associated with the user ID to use for the connection. This must match the password assigned for this user ID *exactly.* If you are using an existing connection, you can specify a NULL string (that is, two quotation marks with no blank space between them ("")) for this parameter.
ExtAuth	This parameter is required, but it is unused in this release. You must enter a 0 for this parameter.

Returns

A Boolean value: True if the connection completed successfully, False otherwise.

Example

The following is an example of the code you would use if there is an existing connection to the application server:

```
Local ApiObject &MYSESSION;

&MYSESSION = GetSession();
&MYSESSION.Connect(1, "EXISTING", "", "", 0);
```

The following is an example connecting to a PeopleSoft system using a Visual Basic program. It first checks for error messages. If there are any errors, the text of the error is displayed to the user.


```
On Error GoTo eMessage

    Dim oSession As Object
    Dim oPSMessage As Object

    Dim oBC As Object

    Set oSession = CreateObject("PeopleSoft.Session")

    oSession.Connect 1, "//PSSOFTO061998:7000", "PTDMO", "PTDMO", 0

    &lsquo;Application Specific Processing

eMessage:
    If (oSession.PSMessages.Count > 0) Then
        For i = 1 To oSession.PSMessages.Count
           Set oPSMessage = oSession.PSMessages.Item(i)
           MsgBox (oPSMessage.Text)
        Next i
    End If
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "System Variables," %Session

Disconnect

Syntax

Disconnect()

Description

The Disconnect method disconnects the session associated with the session object executing the method from that PeopleSoft session.

This method returns a Boolean value: True if successfully disconnected, False otherwise.

API Instantiation Methods

The following is a list of the other methods used with the session object to instantiate an API object such as a Component Interface, a tree structure, and so on.

Component Interface:

- FindCompIntfcs*(partial-_name)* : returns a collection of Component Interfaces based on the partial Component Interface name.
- GetCompIntfc(**COMPINTFC.***componentname)*: returns a Component Interface object.

Portal Registry:

- FindPortalRegistries*(partial_name)*: returns a collection of PortalRegistry objects.
- GetLocalNode(): returns a node object.
- GetNodes(): returns a collection of node objects.
- GetPortalRegistry(): returns a PortalRegistry object.
- GetRemoteNodes(): returns a collection of node objects.
- GetActualRemoteNodes(): returns a collection of remote node objects only.

Query:

- FindQueryDBRecords():returns a reference to a QueryDBRecord collection, filled with zero or more records.
- FindQueries():returns a reference to a Query collection, filled with zero or more queries.
- FindQueriesDateRange*(StartDateString,EndDateString)*: returns a reference to a Query collection, filled with zero or more queries that match the specified date range.
- GetQuery(): returns a query object.
- GetQuerySecurityProfile(): a QuerySecurityProfile object.
- SearchQueryDBRecord*(SearchType,Pattern,CaseSensitive)*: returns a QueryDBRecord collection.
- SearchPrivateQueries*(QueryType, SearchType,Pattern,CaseSensitive*): a query collection.
- SearchPublicQueries(*QueryType, SearchType,Pattern,CaseSensitive*): a query collection.

Search:

- GetSearchIndexes(): returns a SearchIndexes collection.
- GetSearchQuery(): returns a search query object.

Tree:

- GetTree(): returns a tree object.
- GetTreeStructure(): returns a tree structure object.

Session Class Properties

This section describes the Session class properties. The properties are discussed in alphabetical order.

ErrorPending

Description

This property indicates whether there are any error messages pending for the API that is currently running. After this property has been set, it returns True until the PSMessages collection is cleared (or deleted.)

This property is read-only.

Example

```
If &MYSESSION.ErrorPending Then
   &COUNT = &MYSESSION.PSMessages.Count;
   For &I = 1 To &COUNT
      &ERROR = &MYSESSION.PSMessages.Item(&I);
      &TEXT = &ERROR.Text;
      WinMessage("Error text " | &TEXT);
   End-For;

&MYSESSION.PSMessages.DeleteAll();
```

PSMessages

Description

This property returns a reference to a PSMessages collection object. If there are no messages, this property returns an object reference to PSMessages with a count of zero.

This property is read-only.

Example

```
If &SESSION.PSMessages.Count > NULL Then
   /* there are messages do processing */
End-if;
```

See Also

Chapter 38, "Session Class," PSMessages Collection Methods, page 2191

PSMessagesMode

Description

This property is used to determine how messages are output. This property takes either a numeric value or a constant. The default value is 1 (%PSMessages_CollectionOnly).

You must set this property before you check the type of message, that is, you can't check the type of message, then decide how it's displayed.

This property sets the value for the session. You can change modes during a session, such as, if you're starting a Component Interface. However, after you run the Component Interface, you should set the value back. For example:

```
&OldMode = &Session.PSMessageMode;
&Session.PSMessagesMode = 1;
...
&Session.PSMessagesMode = &OldMode;
```

You can specify either a numeric value or a constant. The values are:

Numeric Value	***Constant Value***	***Description***
0	%PSMessages_None	None.
1	%PSMessages_CollectionOnly	PSMessage Collection only (default).
2	%PSMessages_MessageBoxOnly	Message box only.
3	%PSMessages_Both	Both collection and message box.

Note. If you set this property to 0 (%PSMessages_None), *all* messages are ignored. Use this option with caution.

This property is read-write.

Example

```
Local ApiObject &SESSION;

&SESSION = %Session;
&MMODE = &SESSION.PSMessagesMode;
&SESSION.PSMessagesMode = %PSMessages_MessageBoxOnly;
```

RegionalSettings

Description

This property returns a reference to a RegionalSettings collection object.

This property is read-only.

See Also

Chapter 38, "Session Class," Regional Settings Class Properties, page 2198

Repository

Description

This property returns a reference to a Repository object.

See Also

Chapter 7, "API Repository," page 233

SuspendFormatting

Description

This property turns off the PeopleSoft automatic formatting when coercing a string into a field. Formatting is typically necessary for external Component Interface users (such as for a Visual Basic program,) yet should be suspended for a PeopleCode program. You should not typically need to set this value yourself, as it's set appropriately based on where the session object is created.

This property takes a Boolean value: True means formatting is suspended, False means it isn't suspended. If you start your session from a PeopleCode program, the default for this property is True. If you don't start the session from a PeopleCode program, the default is False.

This property is read-write.

TraceSettings

Description

This property returns a reference to a TraceSettings collection object.

This property is read-only.

See Also

Chapter 38, "Session Class," Trace Setting Class Properties, page 2203

WarningPending

Description

This property indicates whether there are any warning messages pending for the API that is currently running.

This property is read-only.

Accessing a PSMessages Collection

Use the PSMessages collection to return all of the messages that occur during the session. The following example finds all the messages listed in the PSMessages collections when the Component Interface Save methods fails. The example assumes that &CI is the Component Interface object reference.

```
If Not &CI.Save() Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;

      /* do message processing */

   End-For;
&COLL.DeleteAll(); /* Delete all messages in queue */
End-if;
```

As you evaluate each message, delete it from the PSMessages collection. Or, delete all the messages in the queue at once using DeleteAll.

If there are multiple errors when saving a Component Interface, all errors are logged to the PSMessages collection, not just the first occurrence of an error.

Considerations for Declaring Collections

You *must* declare all collections for C/C++ and COM as objects, or else you receive errors at runtime.

PSMessages Collection Methods

This section describes the PSMessages collection methods. The methods are discussed in alphabetical order.

DeleteAll

Syntax

DeleteAll()

Description

The DeleteAll method deletes all the PSMessage objects in the PSMessages collection executing the method. You should use this property after you have processed all the errors in the collection. This method also resets the status of ErrorPending to False.

Returns

This method returns a Boolean value: True if all PSMessages in the collection were successfully deleted, False otherwise.

See Also

Chapter 38, "Session Class," ErrorPending, page 2188

DeleteItem

Syntax

`DeleteItem`*`(number)`*

Description

The DeleteItem method deletes the PSMessage object that exists at the *number* position in the PSMessages collection executing the method.

Parameters

Parameter	***Description***
number	Specify the position number in the collection of the PSMessage object that you want to delete. All other items within the collection are renumbered.

Returns

This method returns a Boolean value: True if the PSMessage object was successfully deleted, False otherwise.

Example

```
&ERROR.DeleteItem(&I);
```

First

Syntax

`First``()`

Description

The First method returns the first PSMessage object in the PSMessages collection object executing the method.

Example

```
&ERROR = &PSMESSAGE.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns a PSMessage object that exists at the *number* position in the PSMessages collection executing the method

Parameters

Parameter	*Description*
number	Specify the position number in the collection of the PSMessage object that you want returned.

Returns

A reference to a PSMessage object, NULL otherwise.

Example

```
&PSMSGCOL = &SESSION.PSMessages;
For &I = 1 to &PSMSGCOL.Count;
   &PSMESSAGE = &PSMSGCOL.Item(&I);
   /* do error processing */
End-For;
```


Next

Syntax

```
Next()
```

Description

The Next method returns the next PSMessage object in the PSMessages collection object executing the method. You can use this method only after you have used either the First or Item methods: otherwise the system doesn't know where to start.

PSMessages Collection Property

This section describes the PSMessages collection property.

Count

Description

This property returns the number of PSMessage objects in the PSMessages collection object.

This property is read-only.

Example

```
&COUNT = &PSMESSAGE.Count;
```

PSMessage Class Properties

This section describes the PSMessages class properties. The properties are discussed in alphabetical order.

ExplainText

Description

This property returns the explanation text (as a string) for the error message associated with the PSMessage object. You can use this property in conjunction with the MessageNumber property (which contains the error message number) and the MessageSetNumber property (which contains the error message set number.)

Note. ExplainText should be accessed only when necessary because it requires a second database read or application server roundtrip.

This property is read-only.

MessageNumber

Description

This property returns the error message number (as a number) for the error message associated with the PSMessage object. You can use this property in conjunction with the MessageSetNumber property (which contains the error message set number) and the ExplainText property (which contains text explaining the error.)

This property is read-only.

MessageSetNumber

Description

This property returns the error message set number (as a number) for the error message associated with the PSMessage object. You can use this property in conjunction with the MessageNumber property (which contains the error message number) and the ExplainText property (which contains text explaining the error.)

This property is read-only.

MessageType

Description

This property indicates the type of the message, whether it's from the message catalog, if it's a warning, error, or information error.

This property returns a numeric value. The values are:

Value	***Description***
0	Message is not a message catalog entry or there is no message.
1	Message has a severity of Error.
2	Message has a severity of Warning.
3	Message has a severity of Information.

This property is read-only.

Source

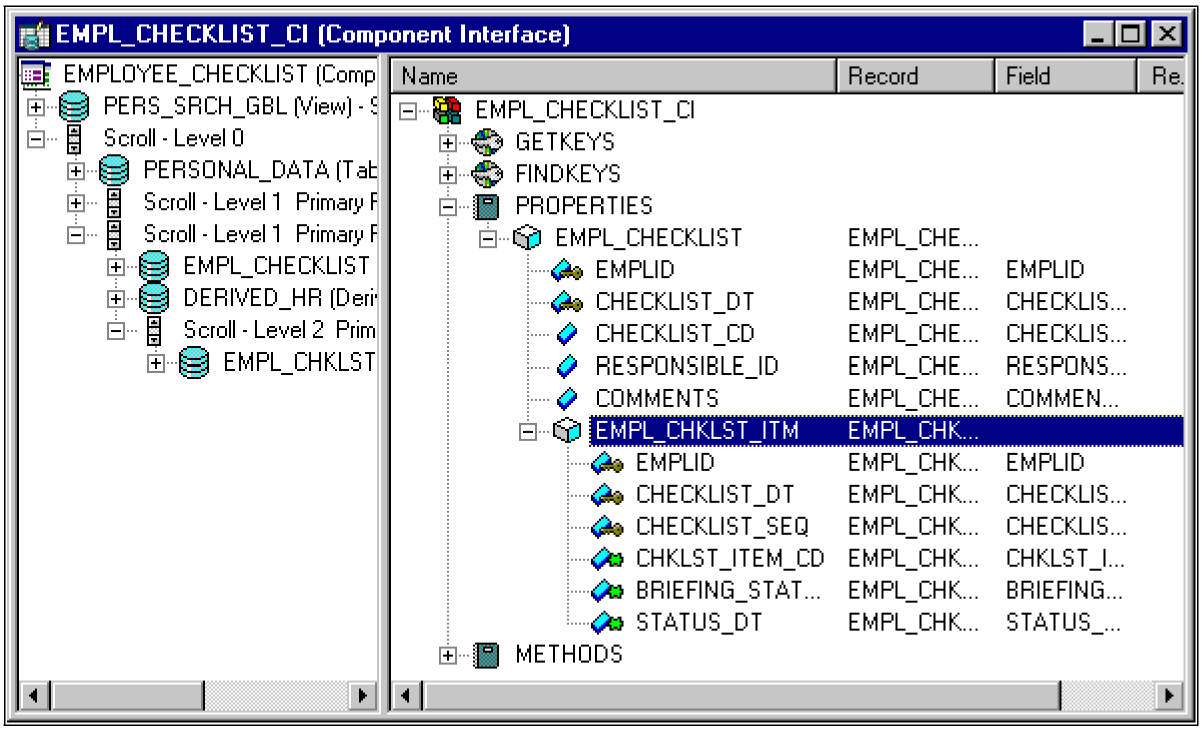
Description

This property returns a string indicating the actual field that's in error. The syntax of the string that's returned is as follows:

```
ComponentInterfaceName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
(Row)]]].PropertyName
```

This string is also returned as part of the Text property.

For example, suppose you had a Component Interface named EMPL_CHKLST_CI.



Sample Component Interface

The following indicates that the first level field, EMPLID, has the error:

```
EMPL_CHKLST_CI.EMPLID
```

The Component Interface EMPL_CHKLST_BC has a data collection (scroll) called EMPL_CHKLST_ITM. Suppose that it has 3 rows, and the STATUS_DT field was in error. The **Source** property would return the following string:

```
EMPL_CHKLST_CI.EMPL_CHKLST_ITM(3).STATUS_DT
```


You can use the Source property in conjunction with the MessageNumber property (which contains the error message number), the MessageSetNumber property (which contains the error message set number), the ExplainText property (which contains text explaining the error), and the Text property (which contains the text of the message.)

This property is read-only.

Example

The following example code finds the error messages and displays them to the user. It finds the field that caused the error and displays that also.

```
Local ApiObject &PSMESSAGE;
Local Boolean &FIND;
Local string &SOURCE, &FIELD, &TEXT;
Local number &START, &LEN, &NSTART, &FLEN;
.
.
.
For &I = 1 to &SESSION.PSMessages.Count;
   &PSMESSAGE = &SESSION.PSMessages.Item(&I);
   /* only display errors, not warnings, to user */
   If &PSMESSAGE.Type = 1 Then
      &TEXT = &PSMESSAGE.Text;
      /* find name of field in error */
      &SOURCE = &PSMESSAGE.Source;
      &LEN = Len(&SOURCE);
      /* find last dot before field */
      &FIND = False;
      &START = Find(".", &SOURCE);
      While Not (&FIND)
         &NSTART = Find(".", &SOURCE, &START + 1);
            If &NSTART = 0 Then
               &FIND = True;
            Else
               &START = &NSTART;
            End-If;
         End-While;
         &FLEN = &LEN - &START;
         &FIELD = Substring(&SOURCE, &START + 1, &FLEN);
         /* display text and field to user */
         Winmessage("You received the following error: " | &TEXT | "For field " |⇒
 &FIELD);
   End-If;
End-For;
```

Text

Description

This property returns the text of the message (as a string) for the error message associated with the PSMessage object. It also includes a contextual string that contains the name of the field that generated the error. The contextual string has the following syntax:

```
{ComponentInterfaceName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
(Row)]]].PropertyName}
```


The contextual string (by itself) is available using the Source property of the PSMessage.

You can use the Text property in conjunction with the MessageNumber property (which contains the error message number), the MessageSetNumber property (which contains the error message set number), and the ExplainText property (which contains text explaining the error.)

This property is read-only.

Example

```
Local ApiObject &MYSESSION, &COL, &ERROR;
Local string &TEXT;
.
.
.
If &MYSESSION.ErrorPending Then
   /* received error */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
         /* do error processing */
      End-For;
End-if;
```

See Also

Chapter 38, "Session Class," Source, page 2196

Regional Settings Class Properties

This section describes the regional settings class properties. The properties are discussed in alphabetical order.

ClientTimeZone

Description

This property specifies the client time zone. This property takes a string value.

This property is read-write.

CurrencyFormat

Description

This property specifies how currency values are displayed. This property takes a number value. The default value is 1.

In the following values, @ represents the character specified with the CurrencySymbol property.

Value	*Description*
0	@1.1
1	1.1@
2	@.1.1
3	1.1.@

This property is read-write.

CurrencySymbol

Description

This property specifies the currency symbol. This property takes a string value. The default value is "$".

This property is read-write.

DateFormat

Description

This property specifies the date format defaults. PeopleTools supports the Short Date Format specification (MDY, DMY, or YMD), and the Date separator setting. When you add a Date field in a record definition, you indicate whether you want to display the century. PeopleTools Date and DateTime fields always show leading zeros for month, day, and year.

This property takes a number value. The values for this property are:

Value	*Description*
0	MDY (default)
1	DMY

Value	Description
2	YMD
3	Default value on system

This property is read-write.

DateSeparator

Description

This property specifies the character used to separate the month, day, year in the date. This property takes a string value. The default value is "/".

This property is read-write.

DecimalSymbol

Description

This property specifies the character used to separate the fractional of the number from the whole. This property takes a string value. The default value for this property is ".".

Note. If you change the value of this property to a comma (,), you should adjust the DigitGroupingSymbol to a different character or some applications may not operate correctly.

This property is read-write.

DigitGroupingSymbol

Description

This property specifies the character used as a thousand or other grouping separator. This property takes a string value. The default value for this property is ",".

This property is read-write.

LanguageCD

Description

This property specifies the language to be used for the transaction. This property takes a string value. The default value for this property is the user's base language.

This property is read-write.

Note. If you change this value dynamically using PeopleCode, this value will *not* change back to the original value when you disconnect. You must manually set it back to the original value.

Example

The following example sets the language code, then at the end of the program, sets it back to the original value.

```
Local ApiObject &SESSION;
Local ApiObject &CI;

/** Get Session ApiObject **/
&SESSION = %Session;

/** Get Message ApiObject **/
&MSG = %IntBroker.GetMessage();
&RS = &MSG.GetRowset();

/** Get Component Interface **/
&CI = &SESSION.GetCompIntfc(compIntfc.NAMES_CI);

&REG_LNG = &SESSION.RegionalSettings.LanguageCd;

For &TRANS_NUM = 1 To &RS.RowCount

/** Store Language Code in Temp Var **/
&LNG = &RS(&TRANS_NUM).PSCAMA.LANGUAGE_CD.Value;

/** Set the Language for this Transaction **/
If &LNG <> &REG_LNG Then
&SESSION.RegionalSettings.LanguageCd = &LNG;
End-If;
/* application specific processing */

&SESSION.RegionalSettings.LanguageCd = &REG_LNG;
&SESSION.Disconnect();
End-For
```


UseLocalTime

Description

This property specifies whether the local time zone of the client should be used to format time fields. This property takes a Boolean value. The default value is True.

This property is read-write.

1159Separator

Description

This property specifies the morning designator. This property takes a string value up to five characters. The default value is AM.

Use the 2359Separator property to specify the afternoon designator.

This property is read-write.

Considerations Using the 1159Separator Property

You cannot use this property as you would other properties. You must use the ObjectGetProperty function for accessing the property. For example:

```
Local ApiObject &SESSION;
Local ApiObject &reg;

/** Get Session ApiObject **/
&SESSION = %Session;
&reg = &SESSION.RegionalSettings;

&REG_LNG = &SESSION.RegionalSettings.LanguageCd;
&MornDesigination = ObjectGetProperty(&reg, "1159Separator");
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ObjectGetProperty

Chapter 38, "Session Class," 2359Separator, page 2203

2359Separator

Description

This property specifies the afternoon designator. This property takes a string value up to five characters. The default value is PM.

Use the 1159Separator property to specify the morning designator.

This property is read-write.

Considerations Using the 2359Separator Property

You cannot use this property as you would other properties. You must use the ObjectGetProperty function for accessing the property. For example:

```
Local ApiObject &SESSION;
Local ApiObject &reg;

/** Get Session ApiObject **/
&SESSION = %Session;
&reg = &SESSION.RegionalSettings;

&REG_LNG = &SESSION.RegionalSettings.LanguageCd;
&AfternoonDesigination = ObjectGetProperty(&reg, "2359Separator");
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ObjectGetProperty

Chapter 38, "Session Class," 1159Separator, page 2202

Trace Setting Class Properties

Each of the following properties relates to the debug pages in PeopleTools.

This section describes the trace setting class properties. The properties are discussed in alphabetical order.

See Also

PeopleTools 8.51 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Using Debug Utilities

API

Description

This property is part of the PeopleSoft API trace and sets the API Information value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

COBOLStmtTimings

Description

This property is part of the SQL trace and sets the COBOL Statement Timings value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

ConnDisRollbackCommit

Description

This property is part of the SQL trace and sets the Connect, disconnect, rollback, and commit value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

DBSpecificCalls

Description

This property is part of the SQL trace and sets the Database API-specific Calls value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

ManagerInfo

Description

This property is part of the SQL trace and sets the Manager Information value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

NetworkServices

Description

This property is part of the SQL trace and sets the Network Services value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

NonSSBs

Description

This property is part of the SQL trace and sets the All Other API Calls besides SSBs value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

OutputUNICODE

Description

This property specifies if the trace file is written in ANSI or UNICODE (for example, Output Unicode Trace File).

This property takes a Boolean value: True means the file will be written in UNICODE, False means it will be ANSI.

This property is read-write.

PCExtFcnCalls

Description

This property is part of the PeopleCode trace and sets the Trace External Function Calls value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCFcnReturnValues

Description

This property is part of the PeopleCode trace and sets the Show Function Return Value value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCFetchedValues

Description

This property is part of the PeopleCode trace and sets the Show Fetched Values value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCIntFcnCalls

Description

This property is part of the PeopleCode trace and sets the Trace Internal Function Calls value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCListProgram

Description

This property is part of the PeopleCode trace and sets the List the Program value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCParameterValues

Description

This property is part of the PeopleCode trace and sets the Show Parameter Values value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCProgramStatements

Description

This property is part of the PeopleCode trace and sets the Trace Each Statement in Program value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCStack

Description

This property is part of the PeopleCode trace and sets the Show Stack value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCStartOfPrograms

Description

This property is part of the PeopleCode trace and sets the Trace Start of Programs value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCTraceProgram

Description

This property is part of the PeopleCode trace and sets the Trace Evaluator Instructions value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCVariableAssignments

Description

This property is part of the PeopleCode trace and sets the Show Assignments to Variables value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

RowFetch

Description

This property is part of the SQL trace and sets the Row Fetch value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

SQLStatement

Description

This property is part of the SQL trace and sets the SQL Statement value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

SQLStatementVariables

Description

This property is part of the SQL trace and sets the SQL Statement Variables value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

SSBs

Description

This property is part of the SQL trace and sets the Set Select Buffers value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

SybBindInfo

Description

This property is part of the SQL trace and sets the Sybase Bind Information value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

SybFetchInfo

Description

This property is part of the SQL trace and sets the Sybase Fetch Information value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

TraceFile

Description

This property specifies the location of the trace file (that is, PeopleTools Trace File value.)

This property takes a string value.

This property is read-write.

Chapter 39

SOAPDoc Class

This chapter provides an overview of SOAPDoc class and discusses the following topics:

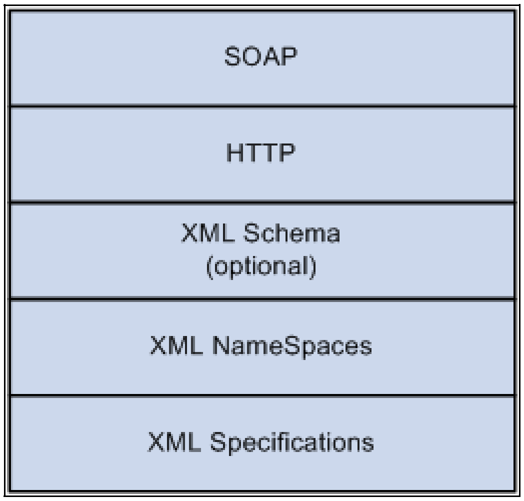
- SOAP message exchange model.
- SOAP message document.
- The SOAPDoc class.
- SOAPDoc object creation.
- The ValidateSOAPDoc method.
- SOAPDoc objects and methods.
- Data type of a SOAPDoc object.
- Scope of a SOAPDoc object.
- SOAPDoc built-in functions.
- SOAPDoc class methods.
- SOAPDoc class properties.

Understanding theSOAPDoc Class

The Simple Object Access Protocol (SOAP) is a lightweight protocol for defining synchronous messages in a distributed Web environment. It's an XML based protocol that can be used in combination with other protocols such as HTTP using URLs as endpoints for a message. SOAP is an application of XML and XML namespaces and optionally uses XML Schemas.

SOAP is an industry standard, defined by the World Wide Web Consortium (W3C), that defines an XML grammar for identifying a method name and the method parameters and for returning the results.

More specific information about SOAP can be found in the W3C specification on SOAP. The SOAP serialization rules can also be found in the complete specifications and aren't outlined in this chapter. Also included in the W3C specification are sections on using SOAP with HTTP, which is also outside the domain of this chapter. You can also send and receive HTTP messages through the Integration Gateway.



SOAP is an application built with XML and HTTP technologies

See Also

http://www.w3.org

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

SOAP Message Exchange Model

SOAP messages are always synchronous. They can implement Request/Response synchronous messages with the response having the output parameters returned in another SOAP document to the sender.

SOAP supports HTTP-POST and HTTP-GET formats. PeopleSoft supports only HTTP, using the Integration Gateway.

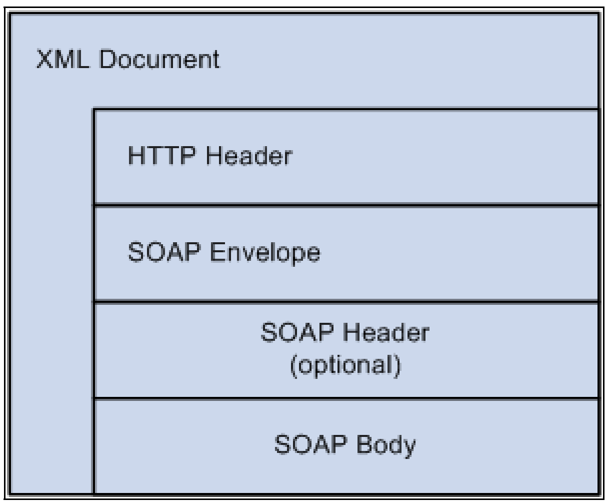
Processing the message requires validation of the mandatory sections of the SOAP XML document and understanding SOAP namespaces.

SOAP Message Document

A SOAP Message is an ordinary XML document that consists of the following parts:

- A mandatory SOAP envelope.
- An optional SOAP header.
- A mandatory SOAP body.

SOAP Message Part	*Description*
SOAP Envelope	Is the top element of the XML document representing the message. It defines the content of the message, as well as the framework of what is in a message, how to process it, who should deal with it and whether it is optional or mandatory.
SOAP Header	Contains header information, and attributes that can be set to encode and further identify the type of processing as well as additional features of the message. This portion is optional.
SOAP Body	Contains the call and response information intended for the recipient of the message. A SOAP Fault element can appear in the body of the SOAP message to carry error or status information.



SOAP parts in an XML document

The following are *required* for a SOAP message:

- Encoded using XML.
- Have a SOAP envelope.
- Have a SOAP body.
- Use the SOAP encoding and envelope namespaces.

The following are optional for a SOAP message:

- SOAP Header.
- XML Schema use.

The following are *invalid* for a SOAP message:

- DTD references.
- XML Processing Instructions.
- XML CData Sections.

The SOAPDoc Class

The SOAPDoc class is a separate class in PeopleCode that works similarly to the XmlDoc and XmlNode classes. Many of the same methods and properties that work with an XmlDoc or XmlNode work with a SOAPDoc. However, there are also methods and properties that don't apply to a SOAPDoc.

The following XmlDoc and XmlNode methods do *not* apply to a SOAPDoc:

- AddCDataSection
- AddProcessInstruction
- CopyToPSFTMessage
- CreateDocumentType
- InsertCDataSection
- InsertProcessInstruction
- LoadIBContent

If a Merchant Profile has been created it can be used with the SOAPDoc class and the Message class. You can use the XmlDoc property to convert the message object to an SOAPDoc object, then use the SOAPDoc methods and properties, as well as the XmlDoc methods and properties that are applicable.

You can only use the message object when the XmlDoc object is structured.

It should be noted that a SOAPDoc object is really an extension of the XmlDoc class. You have the option of getting portions of the SOAPDoc–like the body node–and accessing it or manipulating it as if it were a standard element in an XML document, because in actuality it is.

SOAPDoc Object Creation

There are a few steps to creating a SOAPDoc. The following is an overview of these steps. More detail about the methods and properties mentioned in this section can be found in the reference section of this chapter.

SOAP Header Section

Because the SOAP header is optional, the header is not validated when you use the ValidateSOAPDoc method.

A SOAP header must be added *before* any other parts of the SOAPdoc.

Use the AddHeader method to add a header to a SOAPDoc.

Use the HeaderNode property to access an existing SOAPDoc header.

After you have a reference to the header, you can use XmlDoc methods to read the contents of the header (like any node.)

See Also

Chapter 39, "SOAPDoc Class," AddHeader, page 2224

Chapter 39, "SOAPDoc Class," HeaderNode, page 2233

SOAP Envelope Section

A SOAP envelope is a required portion of the XML Message and is generated automatically using the SOAP schema as specified by SOAP W3C.

A SOAP envelope must be added *before* you add a SOAP body or a SOAP method. This is an optional method. You need to use this method only if you need to set any additional attributes for the Envelope section. The default SOAP schemas are added automatically.

To set additional custom attributes or schemas on the Envelope element, use the AddEnvelope method. This method adds the envelope element tags and the default schemas for either SOAP or UDDI depending on the value of the parameter passed in. The default encoding styles are also set here. Only one envelope section can be set within one PeopleSoft SOAP XmlDoc.

The SOAPDoc adds only a default URL to the envelope. For some UDDI sites, you may need to add a specific schema yourself. To do this, generate the envelope with no schema, then access the envelope section and add the specific schema URL using the AddAttribute method.

See Also

Chapter 39, "SOAPDoc Class," AddEnvelope, page 2220

SOAP Body Section

The body is the XmlDoc where the method is defined. The method AddBody starts the body section of the XML SOAPDoc and must be in the XmlDoc before adding in method sections of the message. Only one body section can be set within one SOAPDoc. This is an optional method and must called only if you need to set any additional attributes for the body section. Otherwise the body section is automatically generated in the XmlDoc.

The SOAP body must be added *after* you add the SOAP envelope, and *before* you add any methods.

See Also

Chapter 39, "SOAPDoc Class," AddBody, page 2219

SOAP Method Section

The AddMethod method adds the name of the method element being processed. Additional attributes for this method can be set using the AddAttribute and InsertAttribute XmlNode methods.

You don't have to specifically add a SOAP body section. The body is automatically added when you call the AddMethod method.

The AddParm method sets the parameters for the parameter element of the method as a name, value pair. Multiple parameters can be set and are used in the order that this method is called. If an XML Schema is defined, types can be used using the AddAttribute XmlNode method. Any number of parameter name-value pairs can be added for a method and are added to the XmlDoc in the order that the AddParm methods are called.

You can add parameters to a method only *after* you add the method. The parameters are added in the order you add them in the PeopleCode program.

See Also

Chapter 39, "SOAPDoc Class," AddMethod, page 2225

Chapter 39, "SOAPDoc Class," AddParm, page 2227

Chapter 46, "XmlDoc Classes," AddAttribute, page 2553

SOAPDoc Section Access

Use the following properties to access the different parts of a SOAPDoc object. All of these properties return an XmlNode object.

- BodyNode
- EnvelopeNode
- MethodNode

After you have access to the portion of the SOAPDoc that you want, use the XmlNode properties and methods to manipulate the SOAPDoc.

Use the XmlDoc property to convert a SOAPDoc object to an XmlDoc object, and vice-versa.

The MethodName property returns the name of the method in the SOAPDoc object.

Note. If you use the MethodName property, you receive only the method name, as a string, not a reference to an XmlNode object that represents the method.

Use the GetParmName and GetParmValue methods to get the name and the values in a SOAPDoc object for each output parameter. You must use the index number of the parameter you want with these methods. You can determine the total number of parameters with the ParmCount property.

See Also

Chapter 39, "SOAPDoc Class," BodyNode, page 2231

Chapter 39, "SOAPDoc Class," EnvelopeNode, page 2231

Chapter 39, "SOAPDoc Class," MethodName, page 2234

Chapter 39, "SOAPDoc Class," MethodNode, page 2234

Chapter 39, "SOAPDoc Class," GetParmValue, page 2228

PeopleCode to Create a SOAPDoc

The following is the order in which the Soap methods *must* be called in order to create a valid SOAPDoc.

```
&SOAPDoc = CreateSOAPDoc(); /* required */
&SOAPDoc.AddEnvelope(0); /* optional */
&SOAPDoc.AddBody(); /* optional */
&SOAPDoc.AddMethod("GetQuote", 1); /* required */
&SOAPDoc.AddParm("symbols", "PSFT");
&SOAPDoc.AddParm("format", "l1c1d1t1");
&OK = &SOAPDoc.ValidateSOAPDoc(); /* optional */
```

SOAPDoc Fault Section

In the body of a SOAP XML response document, you can define a fault section. Generally this is used if in an inbound message has been processed and some sort of error occurred. The text of the message gives further information about the error.

Use the AddFault method to add a fault code and a fault string to the body of a SOAPDoc. Only one fault section can be set within one SOAP XML document.

If you receive a response message that contains a fault, use the FaultCode and FaultString methods to return these values.

See Also

Chapter 39, "SOAPDoc Class," AddFault, page 2222

Chapter 39, "SOAPDoc Class," FaultCode, page 2231

Chapter 39, "SOAPDoc Class," FaultString, page 2232

The ValidateSOAPDoc Method

Use the ValidateSOAPDoc method to validate the SOAP document for correctness as follows:

- Encoded using XML.

- Has SOAP envelope.
- Has SOAP Body.
- Has a SOAP Method.
- Used SOAP encoding and envelope namespaces.

See Also

Chapter 39, "SOAPDoc Class," ValidateSOAPDoc, page 2229

SOAPDoc Objects and Messages

To publish the data of a SOAPDoc, you must first transform the SOAPDoc into an XmlDoc object by using the XmlDoc property. After you do this, you can use the XmlDoc object with the SyncRequestXmlDoc, PublishXmlDoc, or GetXmlDoc functions, to publish the data either synchronously or asynchronously.

All XmlDoc messages must be associated with a message definition. An XmlDoc message must be associated with an *unstructured* message, that is, a message definition that's created without any record definitions.

Use the SOAPDoc class if all of the following are true:

- Your third-party source or target node requires SOAP-compliant messages.
- Your third-party source or target node requires synchronous transactions.
- Your message conforms to the SOAP specification.

See Also

Chapter 46, "XmlDoc Classes," page 2529

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

Data Type of a SOAPDoc Object

SOAPDoc objects are declared as type SOAPDoc. For example,

```
Local SOAPDoc &MyDoc;

Global SOAPDoc &SOAPDoc;
```

Scope of a SOAPDoc Object

A SOAPDoc object can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

Note. Do not extend a SOAPDoc with an application class. This is currently not supported.

SOAPDoc Built-In Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSOAPDoc

SOAPDoc Class Methods

In addition to the methods listed here, most of the XmlDoc and XmlNode class methods can be used with a SOAPDoc object. The methods are discussed in alphabetical order.

See Also

Chapter 46, "XmlDoc Classes," page 2529

AddBody

Syntax

AddBody()

Description

Use the AddBody method to add the body section of the SOAP XML document.

This method is optional when you're creating a SOAP document. Use it only if you must set any additional attributes for the body. Otherwise, the body section is automatically generated in the XML document.

If you're going to add a body section, you must do it *after* you've added the envelope section. If you are going to modify the body section, you must do this *before* you add the method sections of the SOAP XML document by calling the BodyNode property and using XmlNode methods and properties.

Note. Only one body section can be set within one Peoplesoft SOAP XML document.

Parameters

None.

Returns

None.

Example

From the following code:

```
&MyDoc.AddBody();
```

the following XML is created:

```
<SOAP-ENV:Body >

</SOAP-ENV:Body>
```

See Also

Chapter 39, "SOAPDoc Class," AddMethod, page 2225 and Chapter 39, "SOAPDoc Class," AddEnvelope, page 2220

Chapter 46, "XmlDoc Classes," AddAttribute, page 2553

AddEnvelope

Syntax

`AddEnvelope`*`(SOAP_Schema)`*

Description

Use the AddEnvelope method to add the envelope element tags and the default schema. The default encoding styles are also set with this method.

Note. Only one envelope section can be set within one Peoplesoft SOAP XML document.

If you're going to be adding an envelope section, you must do this *before* you add the body section or any methods by calling the EnvelopeNode property and using XmlNode methods and properties.

Parameters

Parameter	***Description***
SOAP_Schema	Specify whether to use the default SOAP schema or the UDDI schema. You can specify either a numeric value or a constant for this property. The values are:

Numeric Value	Constant Value	Description
0	%SOAP_Schema	Use the SOAP schema. Specifying this value adds the following attributes to the SOAP envelope: `xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/⇒ envelope/" xmlns:SOAP-ENV= "http://schemas.xmlsoap.⇒ org/soap/encoding/"`
1	%SOAP_UDDI	Use the UDDI schema. Specifying this value adds the following attribute to the SOAP envelope: `xmlns:SOAP-ENV= "urn:uddi-org:api"`
2	%SOAP_Custom	Specify this value to add custom Namespace schemas to the SOAP envelope instead of the default SOAP schemas. **Note.** You must create the custom schema before you use it. See http://www.w3.org/XML/Schema. You must add the SOAP_ENV URI (as an attribute, using AddAttribute) before adding any other SOAP attributes. **Note.** For highly customized SOAP documents, PeopleSoft recommends using the XmlDoc class and creating the tags yourself.

Returns

None.

Example

The following example uses the default SOAP schema. The following PeopleCode program:

```
&MyDoc.AddEnvelope(%SOAP_Schema);
```

Creates the following XML:

```
<?xml version="1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/encoding/">

</SOAP-ENV:Envelope>
```

The following example creates an envelope and validates it.

The following PeopleCode program:

```
&soapReq = CreateSOAPDoc(); /* required */
/* manually add SOAP XML envelope, header, body, method and parameters */
/* &soapReq.AddHeader(); MUST come before any other section, but is optional */
&soapReq.AddEnvelope(%SOAP_Custom);
&EnvNode = &soapReq.EnvelopeNode;
&AddEnvelopeAttribute = &EnvNode.AddAttribute("xmlns:SOAP-ENV", "urn:MyCustom⇒
Schema.org");
&AddEnvelopeAttribute = &EnvNode.AddAttribute("xmlns:xsi", http://www.w3.org/2001⇒
/XMLSchema-instance);
&AddEnvelopeAttribute = &EnvNode.AddAttribute("xmlns:xsd", http://www.w3.org/2001⇒
/XMLSchema);
&AddEnvelopeAttribute = &EnvNode.AddAttribute("xmlns:soap", http:⇒
//schemas.xmlsoap.org/soap/envelope/);
&soapReq.AddMethod("requestAccessCode", 1);

/* Validate the message */
MessageBox(%MsgStyle_OK, "", 0, 0, "Validate the message");
&OK = &soapReq.ValidateSOAPDoc();
MessageBox(%MsgStyle_OK, "", 0, 0, "Message Validated and &OK = " | &OK);

/* Convert SOAP XML to XML */
MessageBox(%MsgStyle_OK, "", 0, 0, "Sending the message next");

&string = &soapReq.GenXmlString();
MessageBox(%MsgStyle_OK, "", 0, 0, "SOAP XML = " | &string);
```

Creates the following XML:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
xmlns:soap= "http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV= "urn:MyCustomSchema.org">
<SOAP-ENV:Body>
   <requestAccessCode/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

See Also

Chapter 39, "SOAPDoc Class," AddBody, page 2219

AddFault

Syntax

AddFault*(Fault_Code,Fault_String)*

Description

Use the AddFault method to add a fault code, with its corresponding fault string, to a SOAPDoc. Use this method only in a Response SOAP XML Document. The Envelope and Body sections of the XML Document are automatically added if they don't exist.

Parameters

Parameter	*Description*
Fault_Code	Specify the fault code to add, as a string.
Fault_String	Specify the fault string to add, as a sting.

The following fault codes are supported:

Fault Code	*Fault String*	*Description*
100	Version Mismatch	The XML version or SOAP version is not supported by the current implementation.
300	Client error	An error occurred on the client while processing the SOAP document.
400	Server error	An error occurred on the server while processing the SOAP document.

Returns

None.

Example

The following example program:

```
&SOAPDoc = CreateSOAPDoc();
SOAPDoc.AddFault("400", "Server Error");


/* ===> The following statement executes this instance: */
&FaultCode = &SOAPDoc.FaultCode;
&FaultString = &SOAPDoc.FaultString;
&OK = &SOAPDoc.ValidateSOAPDoc();
```

Creates the following XML:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ⇒
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
- <SOAP-ENV:Body>
- <SOAP-ENV:Fault>
  <faultcode>400</faultcode>
  <faultstring>SOAP Server Error</faultstring>
  </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

See Also

Chapter 39, "SOAPDoc Class," FaultCode, page 2231; Chapter 39, "SOAPDoc Class," FaultString, page 2232 and Chapter 39, "SOAPDoc Class," FaultCodeS, page 2232

AddHeader

Syntax

`AddHeader``()`

Description

Use the AddHeader method to add the header section of the SOAP XML document.

This method is optional when you're creating a SOAP document. Use it only if you want to include a header in your SOAP XML document.

If you're going to add a header, you must do it *before* you've added any other sections, such as the envelope or the body.

After you've created the header, you can access it using the HeaderNode property. Once you have a reference to the node, you must use the XmlDoc methods to add specifics to the header section such as encription, as well as any required child nodes.

Note. Only one header section can be set within one Peoplesoft SOAP XML document.

Parameters

None.

Returns

None.

Example

The following test adds a header section into a SOAP XML document where all sections are created explicitly:


```
Local SOAPDoc &Sdoc;
Local XMLNode &Node;

/* ===> Add inputs: */
&Sdoc = CreateSOAPDoc(); /* required */
&Sdoc.AddEnvelope(0); /* optional */
&Sdoc.AddHeader(); /* optional */
&Sdoc.AddBody(); /* optional */
&Sdoc.AddMethod("SOAPXml", 1); /* required */
&Sdoc.AddParm("symbols", "PSFT");

&OK = &Sdoc.ValidateSOAPDoc();

/*Append to TEMP file*/
&Xstring = &Sdoc.GenXmlString();
&myfile = GetFile("cmsDoc.xml", "A");
&myfile.WriteLine(&Xstring);
&myfile.Close();

&METHOD = &Sdoc.MethodName;
&ParmCnt = &Sdoc.ParmCount;
&HdrNode= &Sdoc.HeaderNode;

&HdrName = &HdrNode.NodeName; //returns SOAP-ENV:Header

/* Echo out the Returned Outputs */
out_BI_results("SOAP Header test 1");
out_BI_results(" ");
out_BI_results("Validated: " | &OK);
out_BI_results("Method: " | &METHOD);
out_BI_results("Parm Count: " | &ParmCount);
out_BI_results("Header : " | &HdrName);

out_BI_results("End-of-Test");
```

See Also

Chapter 39, "SOAPDoc Class," HeaderNode, page 2233

AddMethod

Syntax

`AddMethod`*`(MethodName,IsRequest)`*

Description

Use the AddMethod method to set the name of the method element being processed. Additional attributes for the added method can be set using the AddAttribute and InsertAttribute method. Executing AddMethod automatically adds any missing Envelope and Body sections.

Note. Only one method can be set within one Peoplesoft SOAP XML document.

The SOAPDoc method can be used only *after* the envelope and body sections have been added.

Set the parameters for the method using the AddParm method.

Parameters

Parameter	*Description*
MethodName	Specify the name of the method element being added.
IsRequest	Specify if the message is a request message. This parameter takes a numeric value: • 1 if the message is a request message • 0 if the message is a response message

Returns

None.

Example

The following sample program:

```
&MyDoc.AddMethod("GetPrice", 0);
```

Creates the following XML:

```
<GetPrice>

</GetPriceResponse>
```

The following sample program:

```
&MyDoc.AddMethod("GetPrice", 1);
```

Creates the following XML:

```
<GetPrice>

</GetPrice>
```

See Also

Chapter 39, "SOAPDoc Class," AddParm, page 2227; Chapter 39, "SOAPDoc Class," MethodName, page 2234 and Chapter 39, "SOAPDoc Class," MethodNode, page 2234

Chapter 46, "XmlDoc Classes," AddAttribute, page 2553

AddParm

Syntax

AddParm*(ParmName,ParmValue)*

Description

Use the AddParm method to set the parameters for the parameter element of the current method. To set more than one parameter, use the method more than once. The parameters are set and used in the order that this method is called. Any number of parameter name-value pairs can be set for a method and are added to the XML document in the order that this method is called from the PeopleCode program.

Parameters

Parameter	***Description***
ParmName	Specify the name of the parameter as a string.
ParmValue	Specify the value of the parameter as a string.

Returns

None.

Example

The following PeopleCode:

```
&MyDoc.AddParm("Item", "Apples");
```

Creates the following XML:

```
<Item >Apples</Item>
```

See Also

Chapter 39, "SOAPDoc Class," AddMethod, page 2225

GetParmName

Syntax

GetParmName*(Index)*

Description

Use the GetParmName method access the parameter names for a method.

Parameters

Parameter	*Description*
Index	Specify the number of the parameter that you want to access. Values start at 1.

Returns

A string.

Example

```
For &I = 1 to &SOAPDoc.ParmCount

&ParmName = &SOAPDoc.GetParmName(&I);
&ParmValue = &SOAPDoc.GetParmValue(&I);

   /* Do processing */

End-For;
```

See Also

Chapter 39, "SOAPDoc Class," GetParmValue, page 2228 and Chapter 39, "SOAPDoc Class," ParmCount, page 2234

GetParmValue

Syntax

GetParmValue*(Index)*

Description

Use the GetParmValue method to access the parameter values for a method.

Parameters

Parameter	*Description*
Index	Specify the number of the parameter you want to access. Values start at 1.

Returns

A string.

Example

```
For &I = 1 to &SOAPDoc.ParmCount

&ParmName = &SOAPDoc.GetParmName(&I);
&ParmValue = &SOAPDoc.GetParmValue(&I);

   /* Do processing */

End-For;
```

See Also

Chapter 39, "SOAPDoc Class," GetParmName, page 2227 and Chapter 39, "SOAPDoc Class," ParmCount, page 2234

ValidateSOAPDoc

Syntax

ValidateSOAPDoc()

Description

Use the ValidateSOAPDoc method to validate the SOAP document for correctness as follows:

SOAP message is:

- Encoded using XML.
- Has SOAP envelope.
- Has SOAP body.
- Used SOAP encoding and envelope namespaces.

Returns

This method returns either a number or a constant. The values are:

Numeric Value	***Constant Value***	***Description***
0	%SOAP_Valid	The SOAPDoc is valid.
1	%SOAP_NoEnvelope	Missing or invalid envelope section in XML document.
2	%SOAP_InvalidEnvelope	Envelope has improper Namespace used. Should be: SOAP-ENV or SOAP-ENC
3	%SOAP_NoBody	Missing body section in XML document.
4	%SOAP_NoMethod	Missing or invalid method section in XML document.
6	%SOAP_InvalidXml	Invalid XML syntax in SOAPDoc.

Example

```
&Return = &MyDoc.ValidateSOAPDoc();
If &Return <> 0 Then
   /* do error processing */
End-if;
```

See Also

Chapter 39, "SOAPDoc Class," FaultCode, page 2231 and Chapter 39, "SOAPDoc Class," FaultString, page 2232

SOAPDoc Class Properties

In addition to the properties listed here, most of the XmlDoc and XmlNode class properties can be used with a SOAPDoc object. The properties are discussed in alphabetical order.

See Also

Chapter 46, "XmlDoc Classes," page 2529

BodyNode

Description

This property returns a reference to the body section of a SOAPDoc, as an XmlNode.

This property is read-only.

See Also

Chapter 46, "XmlDoc Classes," page 2529

EnvelopeNode

Description

This property returns a reference to the envelope section of a SOAPDoc, as an XmlNode.

This property is read-only.

See Also

Chapter 46, "XmlDoc Classes," page 2529

FaultCode

Description

Use the FaultCode property to return the fault code if the SOAP message has one. Not every message has a fault code. It is returned only when the SOAP method has had an error in processing *and* the SOAP method supports fault codes. This property returns a number.

If the fault code in the message cannot be converted to a number, the value 0 is returned. Use FaultCodeS to get the fault code value as a string.

This property is read-only.

Example

```
Local SOAPDoc &MyDoc;
Local integer &Fault;

&MyDoc = CreateSOAPDoc();
&MyDoc.AddFault("400", "Server Error");

&Fault = &MyDoc.FaultCode;
```


See Also

Chapter 39, "SOAPDoc Class," FaultString, page 2232; Chapter 39, "SOAPDoc Class," FaultCodeS, page 2232 and Chapter 39, "SOAPDoc Class," AddFault, page 2222

FaultCodeS

Description

Use the FaultCodeS property to return the fault code if the SOAP message has one. Not every message has a fault code. It is returned only when the SOAP method has had an error in processing *and* the SOAP method supports fault codes. This property returns a string.

This property is read-only.

Example

```
Local SOAPDoc &MyDoc;
Local string &Fault;

&MyDoc = CreateSOAPDoc();
&MyDoc.AddFault("400", "Server Error");

&Fault = &MyDoc.FaultCodeS;
```

See Also

Chapter 39, "SOAPDoc Class," AddFault, page 2222; Chapter 39, "SOAPDoc Class," FaultCode, page 2231 and Chapter 39, "SOAPDoc Class," FaultString, page 2232

FaultString

Description

Use the FaultString property to return the fault string if the reply has one. Not every SOAP message has a fault string. It is only returned when the SOAP method has had an error in processing *and* the SOAP method supports fault strings.

Example

```
&Fault = &MyDoc.FaultString;
```

See Also

Chapter 39, "SOAPDoc Class," FaultCode, page 2231; Chapter 39, "SOAPDoc Class," FaultCodeS, page 2232 and Chapter 39, "SOAPDoc Class," AddFault, page 2222

HeaderNode

Description

Use the HeaderNode property to return a reference to the header node. After you have a reference, you can use the XmlDoc methods to read and make changes to the node.

This property is read-write.

Example

The following test adds a header section, validates the SOAPDoc, then access the header.

```
Local SOAPDoc &Sdoc;
Local XMLNode &Node;

/* ===> Add inputs: */
&Sdoc = CreateSOAPDoc(); /* required */

&Sdoc.AddHeader(); /* optional */
&Sdoc.AddMethod("SOAPXml", 1); /* required */

&OK = &Sdoc.ValidateSOAPDoc();

/*Append to TEMP file*/
&Xstring = &Sdoc.GenXmlString();
&myfile = GetFile("cmsDoc.xml", "A");
&myfile.WriteLine(&Xstring);
&myfile.Close();

&METHOD = &Sdoc.MethodName;
&ParmCnt = &Sdoc.ParmCount;
&HdrNode= &Sdoc.HeaderNode; //returns SOAP-ENV:Header

&HdrName = &HdrNode.NodeName;

/* Echo out the Returned Outputs */
out_BI_results("SOAP Header test 1");
out_BI_results(" ");
out_BI_results("Validated: " | &OK);
out_BI_results("Method: " | &METHOD);
out_BI_results("Parm Count: " | &ParmCount);
out_BI_results("Header : " | &HdrName);

out_BI_results("End-of-Test");
```

See Also

Chapter 39, "SOAPDoc Class," AddHeader, page 2224

MethodName

Description

This property returns the name of the method of the SOAPDoc, as a string.

If you want a reference to the method that you can work with, as an XmlNode, use the MethodNode property.

This property is read-only.

MethodNode

Description

This property returns a reference to the method section of a SOAPDoc, as an XmlNode.

If you want only the name of the method returned, use the MethodName property.

This property is read-only.

See Also

Chapter 46, "XmlDoc Classes," page 2529

ParmCount

Description

This property returns the total number of parameters for the method section in a SOAP XML document as a number.

This property is read-only.

Example

```
For &I = 1 to &SOAPDoc.ParmCount
   &ParmName = &SOAPDoc.GetParmName(&I);
   &ParmValue = &SOAPDoc.GetParmValue(&I);
   /* do processing */
End-For;
```


XmlDoc

Description

This property transforms a SOAPDoc object to an XmlDoc object. This is necessary only when sending or receiving a response message. You do *not* need to do this conversion to use the XmlDoc or XmlNode methods and properties.

This property is read-write.

Example

The following is an example of sending a SOAP message and receiving the response:

```
Local XmlDoc &request, &response;
Local string &strXml;
Local SOAPDoc &soapReq, &soapRes;

/* create the SOAP XML Document */
&soapReq = CreateSOAPDoc();
&soapReq.AddMethod("GetPrice");
&soapReq.AddParm("Item", "Apples");

/* convert SOAP to XmlDoc */
&request = &soapReq.XmlDoc;

/* Send the Request */
&response = SyncRequestXmlDoc(&request, Message.QE_SOAP_REQ, Node.UNDERDOG);

/* Get the SOAP response from the XmlDoc response */
&soapRes = CreateSOAPDoc();
&soapRes.XmlDoc = &response;
&OK = &soapRes.ValidateSOAPDoc();
&strXml = &soapRes.GenXmlString();
```

SOAPDoc Class Examples

The following are typical example of how you would use a SOAPDoc object.

Creating a SOAP Document

The following is the minimal PeopleCode you need to create a SOAP XML document:

```
Local SOAPDoc &SOAPDoc;

&SOAPDoc = CreateSOAPDoc();
&SOAPDoc.AddMethod("GetPrice");
&SOAPDoc.AddParm("Item", "Apples");
```

The following is an example of creating a SOAP XML document with a simple method.

```
Local SOAPDoc &SOAPDoc;

&SOAPDoc = CreateSOAPDoc();

&SOAPDoc.AddEnvelope(0);
&SOAPDoc.AddBody();
&SOAPDoc.AddParm("symbols", "PSFT");
&SOAPDoc.AddParm("format", "l1c1d1t1");

&OK = &SOAPDoc.ValidateSOAPDoc();
```

Reading an Existing SOAP Document

The following code loads an SOAP XML document from a URL, then finds the method name and parameters to call that method.

```
Local SOAPDoc &SOAPDoc;
Local Number &ParmCount;
Local Boolean &Result;

&SOAPDoc = CreateSOAPDoc();
&SOAPDoc.ParseXmlFromURL("C:\ptdvl\840S2\files\inputSOAP.xml");

&ParmCount = &SOAPDoc.ParmCount;
For &I = 1 to &ParmCount

   &ParmName = &SOAPDoc.GetParmName(&I);
   &ParmValue = &SOAPDoc.GetParmValue(&I);

/* Do processing */
End-for;
```

Using SOAP XML Text

The following shows a SOAP XML Document text for the following pseudo-code:

```
Item = "Apples"
Price = GetPrice(Item);
```

Example of SOAP Request Header

Here's the HTTP Header for a SOAP message. This Host URL is the server where the method defined in the SOAP envelope is processed:

```
POST /GetPrice HTTP/1.1
Host: www.farmersmarket.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction:"GetPrice"
```

This is a simple example of a SOAP XML Document, requesting the price of Apples. It uses the default SOAP schemas. The name of the method is GetPrice with the parameter Item which has a value of "Apples".


```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.SOAP.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.SOAP.org/soap/encoding/"
>
    <SOAP-ENV:Body>
        <m:GetPrice xmnls:m="SomeURI">
        <Item>Apples</Item>
        </m:GetPrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of SOAP HTML Reply Header

Here's the HTTP Header for a SOAP message Reply with a 200 return code that the message was processed OK.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
```

This simple example of a SOAP XML Document reply returns the price of Apples as 34. The SOAP reply always codes the name of the requested method with the Response tag attached to it, GetPriceResponse in this case.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.SOAP.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.SOAP.org/soap/encoding/"
>
   <SOAP-ENV:Body>
       <m:GetPriceResponse xmlns:m="Some-URI">
          <Price>34</Price>
       </m:GetPriceResponse>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Chapter 40

SQL Class

This chapter provides an overview of the SQL class and discusses the following topics:

- Using Record class SQL.
- Creating a SQL definition.
- Binding and executing of SQL statements.
- Fetching from a Select.
- Using Array of Any for bind values.
- Understanding different "Styles" of SELECT.
- Reusing a cursor.
- Understanding SQL objects and Application Engine programs.
- Declaring a SQL object.
- Scope of a SQL object.
- SQL class built-in functions.
- SQL class methods.
- SQL class properties.

See Also

Chapter 40, "SQL Class," Understanding SQL Objects and Application Engine Programs, page 2247

Chapter 34, "Record Class," page 2077

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Understanding SQL Class

You can create SQL definitions in Application Designer. These can be entire SQL programs, or just fragments of SQL statements that you want to re-use. PeopleCode provides the SQL class for accessing these SQL definitions in your PeopleCode program at runtime.

The SQL class provides capability beyond that offered by SQLExec. Unlike SQLExec, which fetches just the first SELECTed row, operations for the SQL class allow iteration over all rows fetched. This can dramatically improve performance if you're doing a million operations and you've set the BulkMode property to True.

A list of input (bind) values, and a list of output variables are supported, as they are in SQLExec. The input and output variables are limited to the same PeopleCode types that can be used with SQLExec, with the addition of a new class called Record.

At runtime, you instantiate a record object from the Record class. A record object is a "one row" instantiation of a record definition.

- When used as an output from an SQL object, each next field in the record is populated with the next column returned.
- When used as an input, several fields in the record are used to replace a single bind marker in the SQL statement. The bind marker for a record describes the type of substitution to be done.

Both records and other PeopleCode types can be mixed in both the output and input.

At runtime, you instantiate a SQL object from the SQL class. The SQL object is loaded by either a constructor for the object, or an explicit Open method call. Optionally, a SQL constructor and the Open method support setting the SQL statement through a string parameter. This capability enables the creation and execution of ad-hoc SQL statements.

The SQL class has a Fetch method for iterating through the rows fetched by a select. A *cursor* is used to control this connection between the runtime SQL object and the database. The cursor is closed automatically when the object goes out of scope. The cursor can be closed before that by using the Close method. The status of the connection is available from the Boolean IsOpen property.

The general status of operations is available from function or method return values or properties. Detailed status is not available, as the operations are designed to terminate on unexpected errors or errors that cannot be reasonably recovered from by application level logic.

Considerations for Extra Spaces

On the DB2 UDB for OS/390 and z/OS platform, when the **zparm** option is set to decimal equals comma (so comma is used as the database decimal separator), extra blanks are added after commas to ensure that they are not mistaken for decimal separators.

Considerations for Case Sensitivity

When processing a SQL statement, the system automatically casts all fieldnames and possibly record names to uppercase when processing a SQL statement. When processing records from a third party, fields that are lowercase get cast into uppercase, which can create a runtime issue on case sensitive platforms.

To prevent this, use the %NoUppercase meta-SQL statement at the beginning of the SQL statement.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "Meta-SQL Elements," %NoUppercase

Using Record Class SQL

There are some SQL type operations that you can do with the Record class, such as INSERT, DELETE and UPDATE. The advantage of using the Record class methods is ease of use, re-use of code, and so on. However, if you're doing many iterations of the same operation (like a million UPDATEs) you should use the SQL object with the BulkMode property set to True.

The SQL object maintains a state (that is, a cursor). Hence, if your database can take advantage of BulkMode, instead of a million operations, the commands are "bulked up" and committed only once. This can improve performance dramatically.

Creating a SQL Definition

You can create SQL definitions in Application Designer, using the SQL editor. These SQL statements can be entire SQL programs, or just fragments that you want to re-use. After you have created a SQL definition, you can use it to populate a SQL object (using FetchSQL, Open, or GetSQL)

You can also create a SQL statement in PeopleCode (using CreateSQL), save it as a SQL definition (StoreSQL), then access it in Application Designer.

See Also

Chapter 40, "SQL Class," Open, page 2252

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using the SQL Editor"

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," FetchSQL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSQL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetSQL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," StoreSQL

Binding and Executing of SQL Statements

The processing of an SQL statement involves a series of steps.

1. The binding process is the replacement of (variable) input values in the statement, in places indicated by bind placeholders.

   The input values are substituted into the SQL statement in place of the bind placeholders. These placeholders have the form "*:number*", or "*%bindop(:number* [, *parm*]. . .)" where the *number* indicates which input value is to be substituted, and the *bindop* and *parm* strings indicate what meta-SQL binding function is to be performed.

   Note. There must be no spaces between the *bindop* and the left parenthesis.

   The following binding meta-SQL functions are used with record objects to substitute various forms of fieldnames and values into the SQL statement. The goal of these binding functions is to enable the writing of SQL and PeopleCode that can manipulate records without dependencies on the exact fields in the records.

   - %DTTM
   - %InsertValues
   - %KeyEqual
   - %KeyEqualNoEffDt
   - %List
   - %OldKeyEqual
   - %Table
   - %UpdatePairs

2. The execution of an SQL statement is the carrying out of the operation of the statement by the database engine.

   This view of SQL statement processing is actually simpler than what actually occurs. In actual practice, the binding occurs in two distinct phases. The database engine (or its supporting routines) is aware of only the latter phase. For some operations, some database engines are able to delay the second stage of binding and the execution of an SQL statement, so the statement can be rebound and re-executed (with different input values). The advantage of this is that these bindings and executions can be accumulated and transmitted across the network to the database server, with several database operations being done in one network operation. This is sometimes referred to as *bulk mode.* Because the network time consumes most database time operations, the performance advantages of bulk mode can be significant.

   However, in bulk mode, individual operations might not be executed immediately by the database engine. The result is that the application might not see errors that arise until later operations are performed.

   SQL SELECT statements are not bound multiple times, rather the retrieved rows are accumulated and sent across the network many rows at a time, also decreasing the effect of network delays.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "Meta-SQL Elements"

Fetching From a Select

Some operations fetch a row of data into a list of output variables. The values of the fields from the select row are assigned in order to the given output variables. If one of these is a reference to a PeopleCode record object, the fields in the record are assigned the successive values from the row of the select, until all the fields are assigned. Assignment then continues with the next output variable, if any. The number of output fields and variables must equal the number of fields in the row of the select.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Using Array of Any for Bind Values or Fetch Results

You can use a parameter of type "Array of Any" in place of a list of bind values or in place of a list of fetch result variables for the Execute, Fetch, and Open methods. This is generally used when you don't know how many values are needed until the code runs.

For example, suppose that you had a SQL definition INSERT_TEST, that dynamically (that is, at runtime) generated the following SQL statement:

```
"INSERT INTO PS_TESTREC (TESTF1, TESTF2, TESTF3, TESTF4, . . . N) VALUES (:1, :2,⇒
 %DateTimeIn(:3), %TextIn(:4)), . . . N";
```

Suppose you have placed the values to be inserted into an Array of Any, say &AAny:

```
&AAny = CreateArrayAny("a", 1, %DateTime, "abcdefg", . . . N);
```

You can execute the insert with the following:

```
&AAny = CreateArrayAny("a", 1, %DateTime, "abcdefg", . . . N);
&SQL = GetSQL(SQL.INSERT_TEST);
While /* Still something to do */
   &SQL.Execute(&AAny);
End-While;
&SQL.Close();
```

Because the Array of Any promotes to absorb any remaining select columns, it must be the last parameter for the SQL object Fetch method or (for results) SQLExec. For binding, it must be the only bind parameter, as it is expected to supply all the bind values needed.

Understanding Different "Styles" of SELECT

Using the SQL object, you have different methods of using a SELECT statement in your code, depending on your application.

The most direct way is with an Open and Fetch. This fetches successive rows, and you can process each row after it's fetched.

The pseudo code for this type of select statement is:

```
&SQL.Open("Select .  .  .", bindpars);
While &SQL.Fetch(resultpars)
   /* process result row */
End-While;
```

Another method is to set up the loop based on something you want to change, and while you still want to change things, continue the loop.

The pseudo code for this type of select statement is:

```
&SQL.Open("Update/Insert/Delete/other.  .  .  :n .  .  .");
/* Bind references, but no bind parameters */
While you_want_to_do_changes
   /* set up for changes */
   &SQL.Execute(bindpars);
End-While;
```

Reusing a Cursor

Reusing a cursor means *compiling* a SQL statement just once, but *executing* it more than once. When a SQL statement is compiled, the database checks the syntax and chooses a query path. By doing this only once, but executing this statement several times, you can obtain an improvement in performance.

In PeopleCode, you can reuse a cursor either by using the ReuseCursor property or by following certain restrictions in your code. In this section, we discuss how to:

- Use the ReuseCursor property.
- Reuse a cursor without the ReuseCursor property.

Using the ReuseCursor Property

If you specify the ReuseCursor property as True, the cursor isn't closed until either the SQL object is explicitly closed or re-opened. This provides greater control over the cursor associated with your SQL object. However, when you set ReuseCursor to True, you're essentially pledging to do the right thing in your program. There are some considerations for how you use this property:

In a Application Engine program, if your program can be restarted, you *must*check for a restart after a checkpoint by testing to see if the SQL object is open after the checkpoint. If it isn't open, that means a restart has happened, and you must reopen the SQL object. In most cases, on checkpoints, your open SQL objects aren't closed, saving the overhead of re-establishing the SQL object after the checkpoint. If the SQL object is open on a select statement at a checkpoint it isn't restored to the open state because you cannot reliably establish the state of the execute-fetch sequence.

In the following example the SQL object is established on a select statement which is executed twice with different bind parameters but is compiled only once. Without the ReuseCursor property the SQL object would be closed after the first fetch cycle completes.


```
Local SQL &Sql;
Local String &Company1 = "CCB";
Local String &Company2 = "CCF";

&Sql = CreateSQL("SELECT A.COMPANY, %Timeout(A.EFFDT) FROM %Table(COMPANY_TBL) A⇒
 WHERE A.COMPANY=:1", &Company1);
/* commenting this out should make the subsequent Execute fail */
&Sql.ReuseCursor = True;

While &Sql.Fetch(&Company1, &Effdt)
   /* processing for the first Company */

End-While;

&Sql.Execute(&Company2);

While &Sql.Fetch(&Company1, &Effdt)
/* processing for the second company */

End-While;

&Sql.Close();
```

Reusing a Cursor Without the ReuseCursor Property

If you want to reuse a cursor in your program, there are several conditions:

- You have to use the SQL object. Only SQL objects retain SQL cursor information.
- For INSERT, DELETE, and UPDATE statements, you automatically reuse the cursor as long as you don't change the SQL statement as part of the binding process. If the SQL statement is textually the same, so only the binds have changed, you get reuse. For example, you won't get reuse if you first bind one kind of record object to %Insert(), then bind another, different kind of record object.
- For SELECTs, you *must*use the meta-SQL shortcuts (%SelectAll, %SelectByKey, or %SelectByKeyEffDt), *and* the CreateSQL or GetSQL functions without supplying any bind or buffer parameters. The bind parameters are supplied in the Execute function. The Fetch parameter must be the fetch buffer, and be the same as the first Execute parameter. You can supply WHERE clauses, ORDER-BY, and so on, on the end of the SQL string containing the meta-SQL.

Note. BulkMode doesn't reuse the cursor in the same way as the SQL object. BulkMode requests that, when the system can reuse the cursor, it also holds all the changes so they can be communicated to the database in batches.

By using one of the forms of the Select meta-SQL, you're guaranteeing the resulting fetched values are all put into one record object (buffer). This means the implementation doesn't have to ask the database for the length and type of each column: the record buffer is already defined. So this sample code uses the SQL object to maintain the state of the connection with the database, and the record object, to maintain a series of fields suitable for database operations.


```
Local SQL &SQL;
Local Record &REC;

&REC = CreateRecord(Record.KP_KPI_DFN);

/* start with select statement, no bind refs, no */
/* bind parameters */

&SQL = CreateSQL("%Selectall(:1) Where SETID = :2 and KPI_ID = :3 and EFFDT = ⇒
(SELECT MAX(EFFDT) FROM PS_KP_KPI_DFN WHERE SETID = :4 AND KPI_ID = :5 AND ⇒
EFFDT <= %DateIn(:6))");
```

Start Loop

```
/* bind and execute the statement */

&SQL.Execute(&REC, &SETID_KPI, &COMPID, &SETID_KPI, &COMPID, &EFFDT);

/* Note record parameter for Fetch statement must be
the same as the first Execute parameter
the results are in this record */

If &SQL.Fetch(&REC) Then

   /*process this record */

End-If;
```

End Loop

```
&SQL.Close(); /* there is no implicit close on a Fetch returning False */
```

The following example comes from a Application Engine program. Because the loop goes in and out of PeopleCode, you must declare the SQL object as Global. This example is in three parts.

1. Program called by Application Engine before the loop:

   ```
   Global SQL &SQL;

   &SQL = CreateSQL("INSERT INTO %Table(MY_WORK) (TRANS , REGISTERWRITE) VALUES (:1, ⇒
   :2)");

   &SQL.BulkMode = True; /* not required for reuse, but will get better performance⇒
    on platforms that support bulk insert */
   ```

2. Program called in the Application Engine loop on SELECT 'X' FROM PSRECDEFN WHERE RECNAME LIKE '%A':

   ```
   Global SQL &SQL;

   &var1 = "X";
   &var2 = "Y";

   &SQL.Execute(&var1, &var2);
   ```

3. Program called by Application Engine after the loop:

   ```
   Global SQL &SQL;

   &SQL.Close();
   ```


Understanding SQL Objects and Application Engine Programs

A global variable won't go out of scope until a Application Engine program finishes. However, all SQL objects are forced closed (that is, the cursor closed) sooner than that. *Any open SQL object is forced closed just before any checkpoint in an Application Engine program.*This is to ensure that the application can be restarted successfully from the checkpoint. After the SQL object is closed, you can reopen the SQL object, or query its properties, (such as Status, IsOpen). The simplest way to avoid unnecessary closing of the SQL object is to set the ReuseCursor property to True. A restartable program should always check that SQL objects are open before using them in steps where it's expected they're open. In the absence of an intervening checkpoint, an open SQL object remains open until the Application Engine program finishes.

Declaring a SQL Object

SQL objects are declared as type SQL. For example:

```
Local SQL &MYSQL;
```

```
Global SQL &MySql = CreateSQL(SQL.MySql);
```

Scope of an SQL Object

An SQL object can be instantiated only from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

Your SQL statements sometimes change the database. When your SQL changes the database, your code should be only in one of the following events:

- SavePreChange
- WorkFlow
- SavePostChange
- Message Subscription
- FieldChange
- Application Engine PeopleCode action

SQL Class Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSQL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," DeleteSQL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," FetchSQL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetSQL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," StoreSQL

SQL Class Methods

In this section, we discuss the SQL class methods. The methods are discussed in alphabetical order.

Close

Syntax

Close()

Description

The Close method closes the SQL object. This terminates any incomplete fetching of select result rows, completes any buffered operations (that is, using BulkMode), and disassociates the SQL object from any SQL statement that was open on it.

After BulkMode operations, the RowsAffected property is not valid.

Parameters

None

Returns

True on successful completion, False if there was a duplicate record error. Any errors associated with buffered operations (that is, using BulkMode), other than duplicate record errors, cause termination.

Example

```
&SQL = CreateSQL("%Delete(:1)");
While /* Still something to do */
   /* Set key field values of &ABS_HIST */
   &SQL.Execute(&ABS_HIST);
End-While;
&SQL.Close();
```

See Also

Chapter 40, "SQL Class," Open, page 2252

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSQL

Execute

Syntax

`Execute``(`*`paramlist`*`)`

Where *paramlist* is an arbitrary-length list of values in the form:

`inval1` `[,` *`inval2`*`] ...`

Description

The Execute method executes the SQL statement of the SQL object. The SQL object must be open and unbound on a delete, insert, or update statement. That is, the CreateSQL, GetSQL, or Open preceding the Execute must have specified a delete, insert, or update statement with bind placeholders and must not have supplied any input values.

The values in *paramlist* are used to bind the SQL statement before it gets executed.

When using the optional BulkMode, the Execute operations may be buffered and are not guaranteed to have been presented to the database until a Close is done. Thus, in BulkMode, errors that arise may not be reported until later operations are done.

Parameters

Parameter	***Description***
paramlist	Specify input values for the SQL string.

Returns

True on successful completion, False for "record not found" and "duplicate record" errors. Any other errors cause termination.

Example

The following example creates a SQL object for inserting. The statement isn't automatically executed when it's created because there aren't any bind variables. The Execute occurs after other processing is finished. The name of the record is passed in as the bind variable in the Execute method.


```
&SQL = CreateSQL("%Insert(:1)");
While /* Still something to do */
   /* Set all the field values of &ABS_HIST.  */
   &SQL.Execute(&ABS_HIST);
End-While;
&SQL.Close();
```

The following example creates two SQL objects, one to be used for fetching, the other for updating the record. The first SQL object selects all the records in the &ABS_HIST record that match &EMPLID. The data is actually retrieved using the Fetch method. After values are set on the record, the update is performed by the Execute.

```
&SQL1 = CreateSQL("%Select(:1) where EMPLID = :2", &ABS_HIST, &EMPLID);
&SQL_UP = CreateSQL("%Update(:1)");
While &SQL1.Fetch(&ABS_HIST);
   /* Set some field values of &ABS_HIST.  */
   &SQL_UP.Execute(&ABS_HIST);
End-While;
&SQL_UP.Close();
```

The following is an example of inserting an array of records:

```
Local SQL &SQL;
Local array of Record &RECS;

/* Set up the array of records.  */
.  .  .

/* Create the SQL object open on an insert */
/* statement, and unbound */
&SQL = CreateSQL("%Insert(:1)");
   /* While the array has something in it */
   While &RECS.Len
   /* Insert the first record of the array, */
   /* and remove it from the array.  */
   &SQL.Execute(&RECS.Shift());
   End-While;
```

See Also

Chapter 40, "SQL Class," Close, page 2248 and Chapter 40, "SQL Class," BulkMode, page 2254

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSQL

Fetch

Syntax

`Fetch`*`(paramlist)`*

Where *paramlist* is an arbitrary-length list of values in the form:

`outvar1` `[`*`, outvar2`*`] ...`

Description

The Fetch method retrieves the next row of data from the SELECT that is open on the SQL object. Any errors result in termination of the PeopleCode program with an error message.

If there are no more rows to fetch, Fetch returns as False, the *outvars* are set to their default PeopleCode values, and the SQL object is automatically closed.

Using Fetch with a closed SQL object is processed the same as when there are no more rows to fetch.

Note. If you want to fetch only a single row, the SQLExec function can perform better, as it fetches only a single row from the server.

The return of Fetch is not optional, that is, you *must* check for the value of the fetch.

Setting Data Fields to Null

This method will *not* set Component Processor data buffer fields to NULL after a row not found fetching error. However, it does set fields that aren't part of the Component Processor data buffers to NULL. It does set work record fields to NULL.

Parameters

Parameter	***Description***
paramlist	Specify output variables from the SQL Select statement.

Returns

The result of Fetch is True if a row was fetched. If there are no more rows to fetch, the result is False.

Example

In the following example, the Fetch method is used first to process a single row, then to process the ABS_HIST record.

```
Local SQL &SQL;
Local Record &ABS_HIST;

&ABS_HIST = CreateRecord(RECORD.ABSENCE_HIST);
&SQL = GetSQL(SQL.SEL27, 15, "Smith");
While &SQL.Fetch(&NAME1, &BIRTH_DT)
   /* Process NAME1, BIRTHDT from the selected row.  */
End-While;

&SQL.Open(SQL.SEL_ABS_HIST, &NAME1, "Smith");
While &SQL.Fetch(&ABS_HIST)
   /* Process ABS_HIST record.  */
```

The following is an example of reading in an array of record objects:

```
Local SQL &SQL;
Local Record &REC;
Local Array of Record &RECS;

/* Get the SQL object open and ready for fetches.  */
&SQL = CreateSQL("%SelectAll(:1) where EMPLID = :2", RECORD.ABSENCE_HIST, &EMPLID);
/* Create the first record.  */
&REC = CreateRecord(RECORD.ABSENCE_HIST);
/* Create an empty array of records.  *
&RECS = CreateArrayRept(&REC, 0);
While &SQL.Fetch(&REC)
   /* We got a record, add it to the array */
   /* and create another.*/
   &RECS.Push(&REC);
   &REC = CreateRecord(RECORD.ABSENCE_HIST);
End-While;
```

See Also

Chapter 40, "SQL Class," Open, page 2252

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetSQL

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SQLExec

Open

Syntax

Open*(sql* [, *paramlist*])

Where *paramlist* is an arbitrary-length list of values in the form:

inval1 [, *inval2*] ...

Description

The Open method associates the *sql* statement with the SQL object. The *sql* parameter can be either:

- A string value giving the SQL statement.
- A reference to a SQL definition in the form **SQL.***sqlname.*

If the SQL object was already open, it is first closed. This terminates any incomplete fetching of select result rows, completes any buffered operations (that is, using BulkMode), and disassociates the SQL object from any SQL statement that was open on it.

Opening and Processing sql

If *sql* is a SELECT statement, it is immediately bound with the *inval* input values and executed. The SQL object should subsequently be the subject of a series of Fetch method calls to retrieve the selected rows. If you want to fetch only a single row, use the SQLExec function instead. If you want to fetch a single row into a PeopleCode record object, use the record Select method.

If *sql* is not a SELECT statement, and either: there are some inval parameters, or there are no bind placeholders in the SQL statement, the statement is immediately bound and executed. This means that there is nothing further to be done with the SQL statement and the IsOpen property of the returned SQL object will be False. In this case, using the SQLExec function would generally be more effective. If you want to delete, insert, or update a record object, use the record Delete, Insert, or Update methods with the record object.

If *sql* is not a SELECT statement, there are no *inval* parameters, *and* there are bind placeholders in the SQL statement, the statement is neither bound nor executed. The resulting SQL object should subsequently be the subject of a series of Execute method calls to affect the desired rows.

Setting Data Fields to Null

This method will *not* set Component Processor data buffer fields to NULL after a row not found fetching error. However, it does set fields that aren't part of the Component Processor data buffers to NULL.

Parameters

Parameter	***Description***
sql	Specify either a SQL string or a reference to a SQL definition in the form **SQL.** *sqlname.*
paramlist	Specify input values for the SQL string.

Returns

None.

Example

Generally, you use the Open method only after you've already gotten a reference to another SQL object. SELECT and SEL_ABS_HIST are the names of the SQL definitions created in Application Designer.


```
Local SQL &SQL;

&SQL = GetSQL(SQL.SELECT);

/* do other processing */

/* get next SQL statement for additional processing */
/* The open automatically closes the previous */
/* SQL statement */

&SQL.Open(SQL.SEL_ABS_HIST, &NAME1, "Smith");
While &SQL.Fetch(&ABS_HIST)
   /* Process ABS_HIST record.  */
End-While;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSQL and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetSQL

SQL Class Properties

In this section, we discuss the SQL class properties. The properties are discussed in alphabetical order.

BulkMode

Description

This property controls the use of bulk mode. Setting this property to True enables the use of bulk mode, and hence removes any guarantee of the synchronous presentation of error status.

Bulk mode is used only with those database connections and operations that support it. Bulk mode can be used with any SQL operation, that is, with INSERTs, DELETEs, or UPDATEs.

If you're using a Application Engine program, and have set this property to True, the rows inserted in BulkMode are committed at the next database commit in your program.

After BulkMode operations, the RowsAffected property is not valid.

The default value for BulkMode is False.

This property is read-write.

Example

The following code is an example of inserting an array of records using bulk mode:


```
Local SQL &SQL;
Local array of Record &RECS;

/* Set up the array of records.  */
   .  .  .
   /* Create the SQL object open on an insert */
   /* statement, and unbound.*/
   &SQL = CreateSQL("%Insert(:1)");
   /* Try for bulk mode.  */
   &SQL.BulkMode = True;
   /* While the array has something in it&mldr; */
   While &RECS.Len
      /* Insert the first record of the array, */
      /* and remove it from the array.  */
      If not &SQL.Execute(&RECS.Shift) then
         /* A duplicate record found, possibly */
         /* in bulk mode.  There is no way to  */
         /* tell which record had the problem. */
         /* One approach to recovery is to fail*/
         /* the transaction and retry it with a*/
         /* process that does only one record  */
         /* at a time, that is, doesn&rsquo;t use    */
         /* bulk mode.*/
         .  .  .;
      End-If;
End-While;
```

See Also

Chapter 40, "SQL Class," RowsAffected, page 2257

Chapter 40, "SQL Class," Binding and Executing of SQL Statements, page 2241

IsOpen

Description

This property returns as True if the SQL object is open on some SQL statement.

This property is read-only.

Example

You might use the following in a Application Engine program, after a checkpoint. MYSELECT is the name of a SQL definition created in Application Designer:

```
If Not &MYSQL.IsOpen Then
   &MYSQL.Open(SQL.MYSELECT);
End-if;
```


LTrim

Description

This property specifies whether values read by the Fetch method are trimmed of blanks on the left, except in the following cases:

- For long columns.
- The record is directly used to buffer the incoming data. In the following example, the values aren't LTrimmed - regardless of the setting of the LTrim property.

```
Local Record &Rec = CreateRecord(Record.QA_TEST);
Local SQL &Sql = CreateSQL("%SelectAll(:1)");
&Sql.LTrim = True; /* the default */
&Sql.Execute(&Rec);
While &Sql.Fetch(&Rec)
   /* do processing */
End-While;
```

This property takes a Boolean value. The default value is True. If this property is set to False, the selected values are not trimmed of blanks on the left.

This property is read-write.

Note. The removal of blanks from the right end of fetched values (RTrimming) still occurs for non-long columns.

Example

```
Local Record &Rec = CreateRecord(Record.QA_TEST);
Local SQL &Sql = CreateSQL(%SelectAll(:1)", &Rec);
&Sql.LTrim = False;
While &Sql.Fetch(&Rec)
   /* do processing */
End-While;
```

ReuseCursor

Description

This property specifies whether the SQL object tries to reuse the open cursor. This property takes a Boolean value, True, to reuse the SQL cursor.

If specified as True, the SQL object won't be closed at checkpoints, and any non-SELECT SQL is restored. In addition the SQL object is not closed after a fetch cycle completes. It remains open ready to be executed, perhaps with different bind parameters.

If you use this property in your application program, you *must* close the SQL object explicitly or it is closed when the object goes out of scope (that is, when the program finishes.)

You must instantiate a SQL object first before you can reuse it.

This property is read-write.

See Also

Chapter 40, "SQL Class," Reusing a Cursor, page 2244

RowsAffected

Description

This property returns the number of rows affected by the last INSERT, UPDATE, or DELETE statement of the SQL object. After BulkMode operations, the RowsAffected property is not valid.

This property is read-only.

Example

The following code is an example that determines if a delete statement actually deleted anything:

```
Local SQL &SQL;

/* Create the SQL object and do the deletion.  */
   &SQL = CreateSQL("Delete from %Table(:1) where EMPLID = :1", RECORD.ABSENCE_⇒
HIST, &EMPLID);
   If &SQL.RowsAffected = 0 Then
   /* We did not delete any rows.  */
   .  .  .
End-If;
```

See Also

Chapter 40, "SQL Class," BulkMode, page 2254

Status

Description

This property returns the status of the last statement executed. You can use either the constant or the numeric value for this property. The values for this property are:

Numeric Value	*Constant Value*	*Description*
0	%SQLStatus_OK	No Errors.
1	%SQLStatus_NotFound	Record not found.

Numeric Value	*Constant Value*	*Description*
2	%SQLStatus_Duplicate	Duplicate Record Found

This property is read-only.

Example

The following example determines what went wrong after an update:

```
Local SQL &SQL;
Local Record &NEWREC, &OLDREC;

/* Create and initialize &OLDREC with the keys of the
record to be updated.  Create and initialize &NEWREC
with the new field values for the record.  */
...;

/* Create and execute the update.  */
&SQL = CreateSQL("%Update(:1, :2)", &NEWREC, &OLDREC);
Evaluate &SQL.Status
When = %SQLStatus_OK
   /* It worked.  */
When = %SQLStatus_NotFound
   /* The OLDREC keys were not found.  */
When = %SQLStatus_Duplicate
   /* The NEWREC keys were already there.  */
End-Evaluate;
```

TraceName

Description

This property enables you to assign a name to a SQL statement that has been created in PeopleCode using CreateSQL. This name is used in the Application Engine timings trace. This property takes a string value.

Note. You *cannot* associate the TraceName property with the execution of a simple SELECT statement created with CreateSQL. This is because the SELECT is executed when the SQL is created, before it has the TraceName assigned. To do this, create a SQL object instead.

If this property isn't set, it defaults to a substring of the SQL statement, indicating the operation and table, (for example, SELECT PS_VOUCHER_LINE.) It may be useful to set TraceName to indicate the origin of the SQL statement.

This property is read-write.

Example

```
&REC = CreateRecord(Record.VOUCHER_LINE);
&SQL = CreateSQL("%selectall(:1) WHERE BUSINESS_UNIT =:2 AND VOUCHER_ID =:3 AND⇒
 VOUCHER_LINE_NUM = :4");
&SQL.TraceName = "AEPROG.SECT1.STEP1.SQL2";
&SQL.Execute(&REC, MATCHING_AET.BUSINESS_UNIT, MATCHING_AET.VOUCHER_ID, &count);
If &SQL.Fetch(&REC) Then
      &count2 = &count2 + 1;
End-If;
```

The previous example would produce the following in the timings trace.

```
SQL Statement                Count    Time      Count    Time      Count    Time      Time
---------------------------- -------- --------- -------- --------- -------- --------- ---------


PeopleCode

AEPROG1.SECT1.STEP1.SQL1                              169     5.371       169     3.083     4.000
AEPROG1.SECT1.STEP1.SQL2                              100     5.371       100     3.083     4.454
                                                                                          --------
                                                                                             8.454
```

Trace Timings Example

You can use this parameter with the Open statement also. The following is an example of how this works:

```
&Sql1 = CreateObject("SQL");
&Sql1.TraceName = "sql1";
&Sql1.Open("Select %FirstRows(1) 'x' FROM psstatus");
&Sql1.Fetch(&Temp);

&Sql2 = CreateObject("SQL");
&Sql2.TraceName = "sql2";
&Sql2.Open("Select 'x' FROM psstatus");
&Sql2.Fetch(&Temp);
```

Value

Description

This property returns the SQL statement associated with the SQL object as a string.

This property is read-only.

Example

To report an error in a SQL definition, including the actual SQL executed, use the Value property to get the text of the SQL statement:


```
Local SQL &SQL;

/* Execute some SQL.  */
&SQL = CreateSQL(SQL.SOMESQL, &EMPLID);
If &SQL.Status = %SQLStatus.NotFound Then
   /* Get the SQL string used.  */
   &SQLSTR = &SQL.Value;
   /* Report the error.  */
   .  .  .;
End-If;
```


Chapter 41

SyncServer Class

This chapter provides an overview of the synchronization server events and the SyncServer class and discusses the following topics:

- Using synchronization server events.
- Using attachment events.
- Scope of a SyncServer object.
- SyncServer class reference.
- Advanced considerations for synchronization.

Important! PeopleSoft Mobile Agent is a deprecated product. The SyncServer class and mobile classes currently exist for backward compatibility only.

Understanding Synchronization Server Events and the SyncServer Class

PeopleSoft Mobile Agent extends the functionality of PeopleSoft Pure Internet Architecture to disconnected mobile devices, allowing users to continue working with their PeopleSoft applications on a laptop computer or personal digital assistant (PDA) while disconnected from the internet or local network.

The mobile events control the PeopleCode programs used with the mobile device. This PeopleCode uses the mobile classes to control the changes an end user makes to the mobile data.

The synchronization server events control the synchronization of mobile data with the PeopleSoft database. The SyncServer class methods and properties are used for data selection and validation during synchronization.

Understanding the Synchronization Events

When the mobile device is synchronized with the PeopleSoft database, the following events occur on the synchronization server. All of these events only fire when any PeopleCode is associated with the event.

Event Name	*Description*
OnConflict	This event only occurs when the conflict resolution for the synchronizable component interface is selected as Custom on the properties dialog box for the definition, on the Synchronization tab.
OnGetDefinition	This event is used for defining Component Interface objects accessed only by a PeopleCode reference.
OnSelect	This event is used for specifying what data should be downloaded (distributed) to the mobile device.
OnValidate	Occurs when any PeopleCode is associated with the event *and* no PeopleCode is associated with the OnValidateSet event. This event is only used for data validation of a medium to large number of objects.
OnValidateSet	This event is used for data validation of a very large number of objects.

In addition, if you create references in the synchronizable Component Interface to be used with attachments, the following events are also available:

Event Name	*Description*
OnGetProperty	This event is used to specify how attachments are accessed and sent to the mobile device.
OnGetPropertyFilter	Occurs before the OnGetProperty event. This event is used for specifying filter rules to an individual attachment.
OnSetProperty	Occurs when the value of an attachment has changed on the mobile device. This event can is to specify which attachments are synchronized from the mobile device to the PeopleSoft database.

Many of the SyncServer class methods and properties are only valid in specific events. The following table lists the event restrictions for the methods and properties. If a method or property is not listed, it does not have restrictions about which event it can be used in.

Method or Property	*Valid Events*
AddReference	Only valid in the PreBuild, PostBuild, and RowInit events.

Method or Property	***Valid Events***
AddReferenceType	Only valid in the OnGetDefinition event.
CheckForChanges	No restrictions, but generally used in the OnSelect event.
SelectAll	No restrictions, but generally used in the OnSelect event.
ConflictAlgorithm	This property is checked after any PeopleCode in the OnConflict event has run.
ConflictStatus	Only valid in the ConflictStatus event.
ExcludeProperty	No event restrictions, but only used with attachments.
ExportLocation	Only valid in the OnGetProperty event.
LastSyncDateTime	Only valid in the OnSelect event.
LastSyncRowCount	Only valid in the OnSelect event.
PropertyValue	Only valid in the OnGetProperty and OnSetProperty events.
SessionID	Only valid in the OnValidateSet event.
SyncIDs	Only valid in the OnSelect event.
TypeID	Only valid in the OnValidateSet event.
validateID	Only valid in the OnValidate event.
validateRowCount	Only valid in the OnValidate event.
validateVersion	Only valid in the OnValidate event.

See Also

Chapter 41, "SyncServer Class," Using the Synchronization Server Events, page 2265

Understanding the SyncServer Class

During synchronization, use the %SyncServer system variable to instantiate and populate a SyncServer object. You can then use this object in one of the synchronization events to specify the details of the current synchronization.

The %SyncServer variable is ignored outside of the synchronization server events, or events called by the synchronization server.

PeopleSoft recommends using the Synchronizing property in events outside of the synchronization server events to separate code that should only be executed during synchronization. For example, suppose you had additional business logic to apply to data coming from a mobile device. You could put the following in one of the Component events that gets called during synchronization, such as SavePreChange.

```
If %SyncServer.Synchronizing Then
   /* more data rules here */
End-If;
```

Considerations Using Synchronization Events and the SyncServer Class

The synchronization events are not required for synchronizing a mobile device. You should only start using the SyncServer class methods and properties in the events after your mobile application is running. The following is the typical development cycle for synchronization:

1. Create and test your component.

   If you are using an existing component, you must still test it to verify that it runs without errors and returns the data you are expecting.

2. Create and test your Component Interface.

   If you are using an existing Component Interface, you must still test it to verify that it runs without errors and returns the data you are expecting.

3. Create and test your mobile pages.

   Once your Component Interface is running, *then and only then* should you make the Component Interface synchronizable and build mobile pages based on it.

   Note. Remember that any changes made to the underlying Component Interface are *not* automatically reflected in any mobile pages built from that Component Interface.

   At this point, you should verify whether the search page associated with the component returns the data you need for your mobile application. If it does, and if you are not dealing with a large number of instances of the Component Interface that need to be synchronized, you are finished. There is no need to go any further and create PeopleCode for synchronization.

4. Refine the conflict resolution for your mobile application.

   The properties for the synchronizable Component Interface specify which version of the data should be used if there is a conflict (device always wins, database always wins, and so on.) Use the OnConflict event to refine the conflict rules if necessary.

5. Refine the data used with your mobile application.

   If the search record associated with the component is not adequate for returning the data you need for your mobile application, create PeopleCode associated with the OnSelect event to better define the data for your mobile device.

   Note. Remember, this event fires when *any* PeopleCode is associated with the event, even if the PeopleCode program is commented out.

   ---

6. Fine tune your mobile application.

   If your mobile application requires a medium to large number of objects, between 100–1000 instances of a Component Interface, you can use the OnValidate event to possibly enhance synchronization performance.

   If your mobile applications requires a large number of objects, over 1000 instances of a Component Interface, you can use the OnValidateSet event to possibly enhance synchronization performance.

   If you need to create references to Component Interface objects in PeopleCode, without defining the relationship in Application Designer, you can use the OnGetDefinition event.

Using the Synchronization Server Events

PeopleSoft Mobile includes synchronization server events to support mobile device synchronization and data distribution. If a component interface is synchronizable, these events exist in its associated PeopleCode.

This section discusses how to use:

- OnConflict
- OnSelect
- OnValidate

See Also

Chapter 28, "Optimization PeopleCode," page 1489

Chapter 41, "SyncServer Class," Advanced Considerations for Synchronization, page 2283

Using OnConflict

The OnConflict event is invoked only when the conflict resolution for the synchronizable component interface is selected as Custom on the properties dialog box for the definition, on the Synchronization tab, and a conflict has occurred.

Property conflicts are generated when the property changed on both the server and the device, and the property value on the server is different than the property value on the device.

OnConflict PeopleCode is run for each conflict to decide whether the device update is accepted or rejected. The OnConflict PeopleCode attached to the level zero object of the Component Interface, if any, is always run after all other OnConflict PeopleCode has run. OnConflict PeopleCode can set the ConflictStatus property on the SyncServer object to %CONFLICTSTATUS_ NOCONFLICT, which causes the conflict to be ignored.

OnConflict PeopleCode can set the ConflictAlgorithm property on the SyncServer object to %CONFLICTALGORITHM_SERVERWINS or %CONFLICTALGORITHM_DEVICEWINS, which causes the device update to be rejected or accepted.

The ConflictAlgorithm property is checked after all OnConflict PeopleCode has run. The ConflictAlgorithm property overrides the ConflictStatus property. If the ConflictAlgorithm property is not set and if the ConflictStatus property is set to %CONFLICTSTATUS_NOCONFLICT for all of the conflicts, the device update is accepted, otherwise the device update is rejected.

Note. If a PeopleCode program exists in OnConflict but is commented out, the synchronization server ignores the associated conflict (that is, behaves as if it were resolved.) This may result in changes being applied to the server database that may be in error.

OnConflict PeopleCode Example: CONFLICTALGORITHM

In this example, if the mobile device is a personal digital assistant (PDA), this code causes changes on the mobile device to always be accepted.

```
If (%SyncServer.ClientPlatform = %MobileDevice_PDA) Then
   %SyncServer.ConflictAlgorithm = %CONFLICTALGORITHM_DEVICEWINS;
End-If;
```

In this example, the program causes the property values from the mobile device owning the property to take precedence.

```
Function isOwner() Returns boolean
   Local boolean &bOwner;
   /* Use user profile information to determine if the
sync user is the owner of this synchronized Component Interface object... */
   Return &bOwner;
End-Function;

If (isOwner()) Then
   /* No conflict, always accept changes from the owner's
device during synchronization... */
   %SyncServer.ConflictAlgorithm = %CONFLICTALGORITHM_DEVICEWINS;
Else
   /* Reject changes from non-owners when a conflicts occur...  */
   %SyncServer.ConflictAlgorithm = %CONFLICTALGORITHM_SERVERWINS;
End-If;
```

Understanding the Merge Algorithm

The merge algorithm is used when the conflict resolution for the synchronizable component interface is selected as Device Winsor Custom on the properties dialog box for the definition, on the Synchronization tab.

When you select Custom, the merge algorithm is executed before the OnConflict PeopleCode is run. While it is possible for the OnConflict PeopleCode to change the outcome of the merge algorithm, it should not do so. The purpose of OnConflict PeopleCode is to decide whether the device update is accepted or rejected.

The following are the merge algorithm rules:

- Property values changed on the device replace property values on the server.
- Property values changed on the server and not on the device are left equal to the server values.
- Objects added on the server are left in place. Objects added on the device are added on the server.

See Also

Chapter 41, "SyncServer Class," ConflictAlgorithm, page 2277

Using OnSelect

The OnSelect event is invoked during synchronization to implement custom data distribution.

To determine which data to send to the mobile device during synchronization, the synchronization server searches for the results of the OnSelect event as it is defined for each Component Interface. In the Component Interface, you can develop PeopleCode in the OnSelect event to select the record rows based on filtering, such as the mobile device owner, person logged in, sales region, and so on.

The OnSelect event uses the SyncServer class property SyncIDs, which is an array of synchronization Ids (from the SYNCID field.)

Using a SQL statement, select the SYNCID field for each row or instance of data to be synchronized, and pass this array of SYNCIDs in the SyncIDsproperty array to the synchronization server for retrieval. The OnSelect event runs only if the Component Interface has a corresponding mobile page on the mobile device.

If the OnSelect event definition contains no PeopleCode, the default search record for the component interface is used to return an unfiltered array of Sync Ids. All data that satisfies the search record for the component interface is synchronized to the mobile device.

If a PeopleCode program exists in OnSelect but is commented out, the PeopleCode is not invoked and nothing is pushed into the SyncIDs array. However, because a PeopleCode program exists, the synchronization server assumes that the SyncIDs array is correctly populated with no SyncID values.

If PeopleCode exists within the OnSelect event definition, the PeopleCode is run and the resulting set of synchronization Ids is used to determine which data is returned to the mobile device. You can define PeopleCode functions containing data distribution rules for reuse or write code that is specific to each Component Interface.

Using the Common Component Function DistributeDataByRules

This function, contained in the FuncLib FUNCLIB_ECMOBIL, can be used in the OnSelect event to filter data to the mobile device.

OnSelect PeopleCode Examples

In this example, the following PeopleCode on an OnSelect event populates the SyncIDs array with all of the sync Ids from the record rows that have an active lead status (LEAD_STATUS), which is a component interface property. This could be used to supply each salesperson with only the leads that are assigned to that salesperson.

```
/* Select Active Leads */
&sql = CreateSQL("SELECT SYNCID FROM PS_RSF_LEAD WHERE LEAD_STATUS = :1", 'A');
While &sql.Fetch(&SyncID)
   %SyncServer.SyncIDs.Push(&SyncID);
End-While;
```

In the following example, the system variable %UserId associates an end user with the PERSON_ID field in the record row. The PeopleCode for the OnSelect event selects all of the synchronization Ids (and associated record rows) that are owned by that user.

```
/* Determine User's Person ID */
SQLExec("SELECT PERSON_ID FROM PSOPRALIAS WHERE OPRID = :1", %UserId, &PersonID);
/* Select Leads with the User's Person ID */
&sql = CreateSQL("SELECT SYNCID FROM PS_RSF_LEAD WHERE PERSON_ID = :1", &PersonID);
While &sql.Fetch(&SyncID)
   %SyncServer.SyncIDs.Push(&SyncID);
End-While;
```

OnSelect and Spidering

Through spidering, rows of data can be required for the mobile device in addition to those selected by the OnSelect event. Suppose that:

- A contact list on a mobile device includes three people: two from the USA states of California and Illinois and one from the Canadian province of Quebec.
- The OnSelect event for state calls for only states or provinces that are in the USA.

The synchronization server retrieves all of the USA states, including California and Illinois, but it also retrieves Quebec because Quebec is already present on the mobile device.

See Also

Chapter 41, "SyncServer Class," SyncIDs, page 2281

Using OnValidate

The OnValidate event is invoked during out-of-date detection during full synchronization. Use it only when you have medium to large set of data, approximately 100–1000 instances of a Component Interface.

Generally, when a Component Interface instance is accessed for synchronization, the Compoent Interface keys are set and the Get method is called on the Component Interface. If you use OnValidate (or OnValidateSet), this call to the Get method is bypassed. This means that the normal initialization events associated with a Compoent Interface (PreBuild, RowInit, and so are) are also bypassed. This can improve performance. This implies that you may want to use OnValidate (or OnValidateSet) if the performance cost of doing the Get (and running the other events) is greater than running SQL to compute the current version of the instance.

Note. OnValidate is not invoked during upload changes or bootstrap synchronization. It is only invoked during update applications and update business data synchronization.

Use the OnValidate event to compute the current version for the synchronized Component Interface object using the SyncServer validateID property, and return the result to the synchronization server using the SyncServer validateVersion property.

If the Component Interface instance does not exist or is not relevant for some other reason (for example, security), not setting a value into version is interpreted as a validate status of No Relevance and the mobile object is removed from the mobile device.

Use the OnValidate event to get the current version for the object and either set that object to be relevant or remove it from the mobile device.

The OnValidate event uses the SyncServer properties validateID and validateVersion.

Considerations Using OnValidate

If you create PeopleCode for this event, you must maintain it. Anytime you expose an additional collection in the Component Interface, or remove a collection, you must change your OnValidate PeopleCode.

If you have any PeopleCode in the OnValidateSet event, even if it is commented out, the OnValidate event does not run.

OnValidate runs on every instance of the Component Interface. This means that any code in this event fires once per Component Interface instance.

OnValidate PeopleCode Example

The following PeopleCode sets validation for this instance of the QE_DEMO_CONTACT component interface.

```
SQLExec("SELECT LASTUPDDTTM FROM %Table(QE_DEMO_CONTACT) WHERE SYNCID=:1",
%SyncServer.validateID, %SyncServer.validateVersion);
```

See Also

Chapter 41, "SyncServer Class," validateID, page 2282

Chapter 41, "SyncServer Class," validateVersion, page 2283

Using Attachment Events

PeopleTools supports attachments for mobile. The attachment consists of header information (such as size, type, description) and the corresponding document object. The attachment events occur when the mobile user synchronizes the mobile device.

OnGetProperty Event

The synchronization server invokes the OnGetProperty event when the value for the attachment property is requested. The specific implementation of this event is defined by the developer, and based on the application requirements. It can be used to specify which attachments are downloaded to the mobile device.

The following is a sample of PeopleCode that could be used in the OnGetProperty event. In the example, the PropertyValue property is used in the GetAttachment function to specify the filename and location of the attachment file on the synchronization server.


```
If QE_MB_CUST_NEED.FTPSITE <> "" And
      QE_MB_CUST_NEED.ATTACHSYSFILENAME <> "" Then
   %SyncServer.PropertyValue = "c:\PSMobile_FileAttach\" |
       QE_MB_CUST_NEED.ATTACHSYSFILENAME;
   &Result = GetAttachment(QE_MB_CUST_NEED.FTPSITE, QE_MB_CUST_⇒
NEED.ATTACHSYSFILENAME,
      %SyncServer.PropertyValue);
End-If;
```

OnGetPropertyFilter Event

The synchronization server invokes the OnGetPropertyFilter event prior to invoking the OnGetProperty event to allow the application to apply filtering rules to an individual attachment.

If the SyncServer ExcludeProperty property is set to True, the synchronization server only sends the header of the attachment to the mobile device. It does not send the contents of the attachment.

Here is a sample of PeopleCode that could be used in the OnGetPropertyFilter event.

```
If ((%SyncServer.ClientPlatform = %MobileDevice_PDA) And
   (SERVICEORDER_DAMAGEDPART.SIZE > &nMaxAttachmentSize)) Then
   %SyncServer.ExcludeProperty = True;
End-If;
```

OnSetProperty Event

The synchronization server invokes the OnSetProperty event when the value for the attachment is modified. The specific implementation of this event is based on application requirements. It can be used to specify whether the attachment is uploaded from the mobile device.

Here is a sample of PeopleCode that could be used in the OnSetProperty event. The PutAttachment function returns the location of the attachment file in PropertyValue.

```
If QE_MB_CUST_NEED.FTPSITE <> "" And
      QE_MB_CUST_NEED.ATTACHSYSFILENAME <> "" Then
     &Result = PutAttachment(QE_MB_CUST_NEED.FTPSITE, QE_MB_CUST_⇒
NEED.ATTACHSYSFILENAME,
      %SyncServer.PropertyValue);
End-If;
```

Using Filtering

Use the CheckForChanges method along with the LastSyncTime and LastSyncRowCount properties to specify whether the component interfaces on the devices are updated. You would use this ability if your component interfaces are relatively static, meaning that no changes will have occurred to that component interface data during most full synchronization operations.

A filter attribute indicates whether a component interface is to be synchronized or not. When the mobile device receives the response, if a full synchronization is in progress, the device looks for the presence of the filter attribute. If it is present, and the value is true, subsequent processing of the component interface is streamlined.

The filtered synchronization processing occurs as follows:

- If a reference list of SyncIDs is present in the response, the synchronization server constructs lists of existing, new and deleted objects. The list of existing objects is ignored. No validate or get requests are issued for these objects. New objects are retrieved from the server, and deleted objects are removed from the device.
- If the select response contains no reference list of SyncIDs, or if all SyncIDs are found to exist on the device, no other processing occurs for that component interface.
- When a component interface is not filtered, the mobile device saves the timestamp and row count information once the processing for that component interface is complete. All component interface objects retain the timestamp and row count from the last time they were fully synchronized.

See Also

Chapter 41, "SyncServer Class," CheckForChanges, page 2273

Chapter 41, "SyncServer Class," LastSyncDateTime, page 2279

Chapter 41, "SyncServer Class," LastSyncRowCount, page 2279

Scope of SyncServer Objects

A SyncServer object can only be instantiated as Local. SyncServer objects are declared using the SyncServer data type. For example:

```
Local SyncServer &MySyncServer;
```

You can also declare and set the SyncServer variable in the same line. For example:

```
Local SyncServer &MySyncServer = %SyncServer;
```

SyncServer Class

The SyncServer object is accessed using the %SyncServer system variable.

SyncServer Class Methods

This section describes the SyncServer class methods.

AddReference

Syntax

AddReference*(SyncID*[, *CIDefnName*])

Description

Use the AddReference method to identify every instance of an ad-hoc referenced component interface that you identified using the AddReferenceType method. Use AddReference in one of the Component build events (PreBuild, PostBuild, RowInit).

Parameters

Parameter	***Description***
SyncID	This is a string containing the synchronization ID (SyncID).
CIDefnName	This is a string containing the name of the referenced Component Interface. Preface this string with the reserved word **CompIntfc.**

Returns

None.

Example

```
%SyncServer.AddReference(&SyncID, CompIntfc.RefCI);
```

See Also

Chapter 41, "SyncServer Class," Advanced Considerations for Synchronization, page 2283

AddReferenceType

Syntax

`AddReferenceType`*`(CIDefnName)`*

Description

Use the AddReferenceType method in the OnGetDefinition event to identify every ad-hoc reference component interfaces. You must have one call for every instance of an an-hoc Component Interface type in the current component interface.

Parameters

Parameter	*Description*
CIDefnName	This is a string containing the name of the referenced Component Interface. Prefix this string with the reserved word **CompIntfc.**

Returns

None.

Example

```
%SyncServer.AddReferencedType(CompIntfc.RefCI);
```

CheckForChanges

Syntax

`CheckForChanges``(`*`SyncDateTime`*`,`*`TotalRowCount`*`[, `*`HasRefList`*`])`

Description

Use the CheckForChanges method to determine if a component interface needs to be synchronized. This method determines the DateTime value of all rows that make up the object, as well as a total count of those rows. If the maximum DateTime value exceeds the device value or if the row counts are different, the component interface is identified as changed and the data in the component interface will be resynchronized.

If neither condition is true, an additional attribute is added to the result element of the response: `filter="true"`.

Note. If you want to filter which component interface objects are updated, you should use CheckForChanges even during a bootstrap, in order to seed the RowCount setting on the device.

If there are rules to apply to the data, they must be applied whenever CheckForChanges returns false. They might also be applied when CheckForChanges returns true, especially if the data is sensitive and the relevance rules are likely to change.

Parameters

Parameter	*Description*
SyncDateTime	Specify the date and time the object was last updated on the server (the maximum update DateTime across all rows that comprise the object), as a DateTime value.

Parameter	Description
TotalRowCount	Specify the number of rows in the object on the server, as a number.
HasRefList	Specify if the response must contain a reference list even when the component interface is being filtered. This parameter takes a Boolean value. The default value is false. If you specify true for the value, the relevance rules in the OnSelect event must always be applied, even when the CheckForChanges method returns True, to ensure that the rule or authorization changes take effect immediately. If the value of this parameter is false, and if CheckForChanges returns true (which means this component interface will not be processed), then no reference list is included in the response. This means the synchronization processor skips the get and delete processing of this component interface, and the device data is left as-is.

Returns

A Boolean value: true, the device data is identical to the existing data and so the component interface should not be processed, false indicates that the data has changed and the component interface should be resynchronized.

Example

In the following example, the component interface is automatically filtered and no processing is done.

```
%SyncServer.CheckForChanges(%SyncServer.LastSyncTime, %SyncServer.LastSyncRow⇒
Count);
```

In this example, applying the relevance rules is complex and slow, and you only want to do it when changes have occurred, so the code includes false as the third parameter for CheckForChanges. This example code would go in an OnSelect event.

```
Local datetime &SyncDttm;
Local number &RowCount;

/* The following logic enables filtering based on whether any data has changed in⇒
 the underlying table. */
SQLExec("SELECT MAX(SYNCDTTM), COUNT(*) FROM PS_LEVEL0", &SyncDttm, &RowCount);

If %SyncServer.CheckForChanges(&SyncDttm, &RowCount, False) = False Then
   /* Add logic here to populate %SyncServer.SyncIDs based on relevance rules. */
   /* code here */
End-If;
```

In the following example, the logic filters according to a customer defined synchronization frequency (in all days.) This type of code would go in the OnSelect event.

```
Local datetime &DateTime;
Local number &SyncFrequency;

If All(%SyncServer.LastSyncDateTime) Then
   /* Not bootstrap sync.  Determine the DateTime when we should next perform a⇒
 full sync. */
   SQLExec("SELECT FREQUENCY FROM PS_SYNC_FREQ_TBL WHERE CINAME = :1", Comp⇒
Intfc.MYCI, &SyncFrequency);
   &DateTime = AddToDateTime(%SyncServer.LastSyncDateTime, 0, 0, &SyncFrequency,⇒
 0, 0, 0);
Else
   /* Bootstrap sync.  Seed &DateTime so the if-condition below will be true. */
   &DateTime = %Datetime;
End-If;

If &DateTime <= %Datetime Then
   /* Time for a full sync.  No need to call CheckForChanges, because
SyncServer will always use current date-time in the response,
and the row count is irrelevant. */
Else
   /* Time to filter.  Call CheckForChanges with request value to force it. */
   %SyncServer.CheckForChanges(%SyncServer.LastSyncDateTime, 0);
End-If;

/* This CI has no relevance rules.  Tell SyncServer to select all instances. */
%SyncServer.SelectAll();
```

See Also

Chapter 41, "SyncServer Class," LastSyncDateTime, page 2279 and Chapter 41, "SyncServer Class," LastSyncRowCount, page 2279

Notify

Syntax

Notify(*message_text,severity* [, *object_type*, *object_id*])

Description

Use this method to pass an error message during synchronization.

Parameters

Parameter	***Description***
message_text	Specify the message text.
severity	Specify the severity of the message.

Parameter	*Description*
object_type	Specify the object type.
object_id	Specify the object ID.

Returns

None.

SelectAll

Syntax

`SelectAll``()`

Description

Use the SelectAll method to perform the default select processing. If this method is called in an OnSelect event, all instances of the component interface are added to the reference list in the select response. This means that the SyncIDs array is ignored, even if it is not empty.

Warning! If the SelectAll method is not called in the OnSelect event, and if CheckForChanges does not suppress the reference list, the response has a reference list containing all elements of the SyncIDs array. If the array is empty, the reference list is also empty, therefore *all* instances of that component interface are removed from the device.

Parameters

None.

Returns

None.

See Also

Chapter 41, "SyncServer Class," CheckForChanges, page 2273

SyncServer Class Properties

This section describes the SyncServer class properties.

ClientPlatform

Description

This property specifies the type of mobile device.

Value	*Description*
%MobileDevice_PDA	A PDA.
%MobileDevice_Laptop	A laptop computer.

This property is read-write.

ConflictAlgorithm

Description

This property is used to cause the device update to be rejected or accepted. This property is checked after all OnConflict PeopleCode has run. This property overrides the ConflictStatus property.

Value	*Description*
%CONFLICTALGORITHM_ SERVERWINS	The server always wins, which causes the device update to be rejected.
%CONFLICTALGORITHM_ DEVICEWINS	The mobile device always wins, which causes the device update to be accepted.

This property is read-write.

ConflictStatus

Description

This property sets the conflict status. This property is valid only with the OnConflict event.

Value	*Description*
%CONFLICTSTATUS_ NOCONFLICT	Remove from the list of conflicts.
%CONFLICTSTATUS_ CONFLICT	Do not remove from the list of conflicts.

This property is read-write.

ExcludeProperty

Description

This property indicates to synchronization server that the contents of an attachment should *not* be downloaded to the mobile device. (The header is still downloaded.)

This property is only used with attachments.

This property is read-write.

Example

```
If ((%SyncServer.ClientPlatform = %MobileDevice_PDA) And
   (SERVICEORDER_DAMAGEDPART.SIZE > &nMaxAttachmentSize)) Then
   %SyncServer.ExcludeProperty = True;
End-If;
```

ExportLocation

Description

This property allows a developer to instruct the mobile device to export a copy of an attachment to a specific file on the device's disk during synchronization.

You can use this property to automatically extract product images (JPEGs) to a directory on the mobile device to make them accessible from HTML image tags included in HTML that is generated as part of an HTML Area.

This property takes a value of a relative or fully qualified path and filename. If you want to put files into the Res directory (all graphics must be in this directory to display correctly) you could use the following PeopleCode:

```
%SyncServer.ExportLocation = "..\res\filename.ext";
```

This property is used in the OnGetProperty event for an attachment reference file.

This property is read-write.

IsReferenceUnresolved

Description

When two or more new objects are uploaded during a synchronization, and they contain references to each other, it can create a deadlock condition on the server, since the references cannot be resolved in the database yet, and therefore neither of the objects can be successfully saved. To break the deadlock, SyncServer performs a tentative save on one of the objects without attempting to resolve any of it's references. This makes it possible for the other object's reference to be resolved, and it can be permanently saved. Once that is completed, the first object's reference can also now be resolved, so a second save is performed to add the appropriate key values from the resolved reference. The purpose of this enhancement is to provide a new

This property indicates whether or not a tentative save is in progress, and whether the object being processed has had it's references resolved.

This property returns a Boolean value: true if a tentative save is in progress, false otherwise.

This property is read-only.

LastSyncDateTime

Description

This property returns the datetime from the last time this component interface was synchronized, as a DateTime value. This property is used only in the OnSelect event.

This property is read-only.

See Also

Chapter 41, "SyncServer Class," CheckForChanges, page 2273 and Chapter 41, "SyncServer Class," LastSyncRowCount, page 2279

LastSyncRowCount

Description

This property returns the number of rows that were available in this component interface the last time it was synchronized, as a number. This property is used only in the OnSelect event.

This property is read-only.

See Also

Chapter 41, "SyncServer Class," LastSyncDateTime, page 2279 and Chapter 41, "SyncServer Class," CheckForChanges, page 2273

PropertyValue

Description

This property provides the path and filename of the attachment file on the synchronization server. It is used with the GetAttachment and PutAttachment functions.

In the OnGetProperty event, set PropertyValue in the GetAttachment function to the filename and location of the attachment file.

In the OnSetProperty event, the PutAttachment function returns the location of the attachment file in PropertyValue.

This property is read-write.

Example

In the code example below, the second If statement checks for whether the attachment was successfully retrieved, prior to setting the value for PropertyValue.

```
If QE_MB_CUST_NEED.FTPSITE <> "" And
      QE_MB_CUST_NEED.ATTACHSYSFILENAME <> "" Then
   %SyncServer.PropertyValue = "c:\PSMobile_FileAttach\" |
       QE_MB_CUST_NEED.ATTACHSYSFILENAME;
   &Result = GetAttachment(QE_MB_CUST_NEED.FTPSITE, QE_MB_CUST_⇒
NEED.ATTACHSYSFILENAME,
      %SyncServer.PropertyValue);
End-If;
```

SessionID

Description

This property returns the session ID. This is not unique for a device. It is used for internal chunking in OnValidateSet, and can only be used with this event.

This property is read-only.

See Also

Chapter 41, "SyncServer Class," Using OnValidateSet, page 2284

Example

The following PeopleCode sets validation for batches of instances of the QE_DEMO_CONTACT Component Interface.

```
SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=0 FROM
PSSYNCSERVERVAL A, %Table(QE_DEMO_CONTACT) B WHERE A.SYNCSESSIONID =
:1 AND A.SYNCTYPEID = :2 AND (A.SYNCID = B.SYNCID AND A.SYNCDTTM >=
B.LASTUPDDTTM)", %SyncServer.SessionID, %SyncServer.TypeID);

SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=1 FROM
PSSYNCSERVERVAL A, %Table(QE_DEMO_CONTACT) B WHERE A.SYNCSESSIONID =
:1 AND A.SYNCTYPEID = :2 AND (A.SYNCID = B.SYNCID AND A.SYNCDTTM <⇒
 B.LASTUPDDTTM)',
%SyncServer.SessionID, %SyncServer.TypeID);
```

Synchronizing

Description

This property is true during synchronization and false otherwise. Use this to perform (or omit) tasks during synchronization. For example, you might limit the number of rows processed in a child scroll during synchronization.

This property is read-only.

SyncIDs

Description

This property is an array containing the synchronization Ids. Populate the SyncIDs array with the sync Ids of the synchronizable component interface objects that you want to be synchronized with the mobile objects on the mobile device. Use the array class Push method to add a SyncID to the SyncIDs array.

The SyncID array is used as a list of sync Ids for instances that satisfy a set of relevance criteria, such as only synchronize Customer objects that are associated to one salesperson. A SQL object that runs a select on a view might be used to calculate the set of sync Ids for Customer objects that are assigned to that salesperson.

This property can be used only with the OnSelect event.

This property is read-write.

See Also

Chapter 8, "Array Class," page 257

TypeID

Description

This property contains the sync ID for the current component interface type (not just an instance, but the entire type.) This property is used for internal chunking in OnValidateSet, and can only be used with this event.

This property is read-only.

Example

The following PeopleCode sets validation for batches of instances of the QE_DEMO_CONTACT component interface.

```
SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=0 FROM
PSSYNCSERVERVAL A, %Table(QE_DEMO_CONTACT) B WHERE A.SYNCSESSIONID =
:1 AND A.SYNCTYPEID = :2 AND (A.SYNCID = B.SYNCID AND A.SYNCDTTM >=
B.LASTUPDDTTM)", %SyncServer.SessionID, %SyncServer.TypeID);
```

```
SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=1 FROM
PSSYNCSERVERVAL A, %Table(QE_DEMO_CONTACT) B WHERE A.SYNCSESSIONID =
:1 AND A.SYNCTYPEID = :2 AND (A.SYNCID = B.SYNCID AND A.SYNCDTTM <⇒
 B.LASTUPDDTTM)', %SyncServer.SessionID, %SyncServer.TypeID);
```

See Also

Chapter 41, "SyncServer Class," Using OnValidateSet, page 2284

validateID

Description

This property contains the sync ID for the only current *instance* of the Component Interface. It is created using the SYNCID field for that component interface (that field is set in the record properties for that Component Interface.)

This property can be used only with the OnValidate event.

This property is read-write.

validateRowCount

Description

This property returns the total number of rows in a hierarchical synchronizable Component Interface object.

For example, a service order with 10 lines has a validateRowCount value of 11. The property default is 1.

This property can be used only with the OnValidate event.

This property is read-write.

validateVersion

Description

This property contains the version for the instance of the Component Interface. Set validateVersion using the SYNCDTTM field. If the component interface object does not exist, or if the object is not to be used for some other reason (such as security), then do not set the validateVersion property.

This property can be used only with the OnValidate event.

ValidateVersion is initially set to the device version (that is, `max(syncdttm)` across all rows that make up the object). When leaving OnValidate, ValidateVersion must be reset to reflect the current version of the object.

If you want the object removed from the device, don't set the validateVersion property.

This property is read-write.

Example

The following code example forces an object to be re-synched:

```
%SyncServer.ValidateVersion = %CurrentDateTime;
```

If you know the object hasn't changed and want to keep it from being re-synced, use the following code (assuming the VersionRowCount property has also been set correctly):

```
%SyncServer.ValidateVersion = %SyncServer.ValidateVersion;
```

Advanced Considerations for Synchronization

After you mobile application is running without errors, you might decide to further tune your application. Two events enable you to do this in specific circumstances.

This section discusses the following:

- Using OnValidateSet
- Using OnGetDefinition

Using OnValidateSet

You should only use OnValidateSet when you have a large number of Component Interface instances to be synchronized, more than 1000.

Before you try to create an OnValidateSet PeopleCode program, you should first create OnValidate PeopleCode. This enables you to determine the shape of the Component Interface instance, the number of child rows and collections. Having a thorough understanding of this helps you create the OnValidateSet PeopleCode.

Though the OnValidateSet PeopleCode may improve performance, there is additional maintenance with it. Every time you add an exposed collection, or remove one, you must change your PeopleCode program to reflect the new data structure.

Note. If you have any PeopleCode in OnValidateSet, including a program that has been commented out, the OnValidate event does not run.

When this event occurs, a table on the synchronization server (PSSYNCSERVERAL) is populated with all of the synchronization Ids for the component type. They table is keyed by Session ID (SYNCSESSIONID field) and Type ID (SYNCTYPEID field).

The status of the row is tracked in the status column (SYNCSTATUS field). All of the rows in the table begin with a status of 3, meaning that the corresponding Component Interface instance should be purged from the synchronization server. The other valid values for the status field are:

Status	***Description***
0	Current. This causes the corresponding Component Interface instance to not be synchronized.
1	Out of date. This causes a Get to be called on the corresponding Component Interface instance, and the rest of the event flow to be joined.

The OnValidateSet event uses the SyncServer properties SessionID and TypeID. These should be populated by values from the SYNCSESSIONID and SYNCTYPEID fields, respectively.

Considerations Using OnValidateSet

If you set breakpoints in a Component Interface, or do a trace , you will see OnValidateSet running multiple times. This happens when the synchronization server sends a request to an application server for OnValidateSet to run and does not receive a response soon afterward. Then the synchronization server sends out another request. In theory, OnValidateSet only runs once per Component Interface Type. In practice, more than one OnValidateSet request runs concurrently in certain environments.

OnValidateSet PeopleCode Example

The following PeopleCode sets validation for batches of instances of the QE_DEMO_CONTACT component interface.


```
SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=0
WHERE SYNCSESSIONID =:1 AND SYNCTYPEID = :2
AND EXISTS (SELECT 'X' FROM %Table(QE_DEMO_CONTACT) B
WHERE PSSYNCSERVERVAL.SYNCID = B.SYNCID
AND PSSYNCSERVERVAL.SYNCDTTM >= B.LASTUPDDTTM)",
%SyncServer.SessionID, %SyncServer.TypeID);

SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=1
WHERE SYNCSESSIONID =:1 AND SYNCTYPEID = :2
AND EXISTS (SELECT 'X' FROM %Table(QE_DEMO_CONTACT) B
WHERE PSSYNCSERVERVAL.SYNCID = B.SYNCID
AND PSSYNCSERVERVAL.SYNCDTTM < B.LASTUPDDTTM)",
%SyncServer.SessionID, %SyncServer.TypeID);
```

Using OnGetDefinition

Using the AddReference and AddReferenceType methods, you can create ad-hoc references to Component Interfaces using PeopleCode, instead of defining the reference and the relationship in Application Designer.

Use the OnGetDefinition event when you want to access the ad-hoc references specified with the AddReference method in the component events. In this event, you must specify every instance of an ad-hoc reference. The referenced Component Interfaces are defined by their SyncIDs. The referenced Component Interfaces objects can reference other referenced Component Interface objects.

When to Use OnGetDefinition

You should only consider using ad-hoc references when you want to dynamically establish read-only references of static data. Using these methods, you are not establishing a relationship between the Component Interfaces. You cannot navigate these references. You cannot make any changes to these instances.

OnGetDefinition PeopleCode Example

In the OnGetDefinition event, have one call to the AddReferenceType method for every referenced Component Interface type in the current Component Interface.

```
%SyncServer.AddReferencedType(CompIntfc.RefCI);
```

In one of the Component build events (PreBuild, PostBuild, RowInit) you must also include a call to the AddReference method for every instance (identified by the SyncID) of the referenced Component Interface.

```
%SyncServer.AddReference(&SyncID, CompIntfc.RefCI);
```


Chapter 42

TransformData Class

This chapter provides an overview of the TransformData class and discusses:

- Creating a TransformData object.
- TransformData class properties.

Understanding the TransformData Class

When Integration Broker invokes a Application Engine transform program, it inserts the message content into a PeopleCode system variable, %TransformData, which remains in scope throughout the Application Engine program. Each program step can access the variable in turn and modify its content, which then becomes available to the next step.

In the Application Engine program, XSLT steps and PeopleCode steps access %TransformData differently:

- In XSLT, the data is automatically made available to your program. The XSLT program is literally a presentation of the output structure and data, which includes XSL tags that reference, process, and incorporate the input data into the output structure. There's no need to explicitly refer to %TransformData, which automatically receives the interpreted result of the XSLT step.
- In PeopleCode, use the PeopleCode %TransformData system variable to access the TransformData object. You access the XML data as the XmlDoc property of the TransformData object, which you then assign to an XmlDoc object and process normally. Because the XmlDoc object is a reference to the data portion of %TransformData, your modifications are automatically passed back to the system variable.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Applying Filtering, Transformation and Translation"

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "System Variables," %TransformData

Creating a TransformData Object

A TransformData object is returned from the system variable %TransformData. You cannot create a TransformData object directly.

When Integration Broker invokes a Application Engine transform program, it inserts the message content into the %TransformData system variable.

TransformData Class Properties

In this section, the TransformData class properties are presented in alphabetical order.

DestMsgName

Description

This property returns the name of the message at the receiving node as a string.

This property is read-only.

DestMsgVersion

Description

This property returns the name of the message version at the receiving node as a string.

This property is read-only.

DestNode

Description

This property returns the name of the node receiving the message as a string.

This property is read-only.

RejectTransform

Description

Use this property to terminate asynchronous transactions (asynchronous physical transformations only).

Use the following system constant to set this property: %IB_Transform_Rejected.

If this property is set within a transform program for an inbound asynchronous transaction, the result will be that no subscription contract is created. On the Service Operation Monitor Details page for the transaction, an informational message will be part of the error message link indicating that the transaction was terminated.

If this property is set within a transform program for an outbound asynchronous transaction, the publication contract status will be updated to done. The outbound message will not be sent. In addition, on the Service Operation Monitor Details page for the transaction, an informational message will be part of the error message link indicating that the transaction was terminated.

This property is read-write.

RoutingDefnName

Description

This property returns the routing definition name as a string. You can use this property to retrieve the connector's routing properties.

This property is read-only.

See Also

Chapter 25, "Message Classes," LoadConnectorPropFromRouting, page 1359

SourceMsgName

Description

This property returns the name of the message at the sending node as a string.

This property is read-only.

SourceMsgVersion

Description

This property returns the name of the message version at the sending node as a string.

This property is read-only.

SourceNode

Description

This property returns the name of the node sending the message as a string.

This property is read-only.

Status

Description

Use this property to communicate the success or failure of the transform program step to Integration Broker. Use the following values to set this property:

Integer Value	*System Constant*	*Description*
0 (default)	—	Indicates success.
1	—	Indicates that the message failed a filtering step.
2	%IB_Transform_Error	Indicates that an error occurred.

This property is read-write.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker, "Applying Filtering, Transformation and Translation"

XmlDoc

Description

This property contains the XML message data.

You can assign this to an XmlDoc object and process the data using the XmlDoc class methods and properties.

This property is read-write.

See Also

Chapter 46, "XmlDoc Classes," page 2529

Chapter 43

Tree Classes

This chapter provides an overview of tree classes and discusses the following topics:

- Relationships between different tree classes.
- Collections in the tree classes.
- Error handling with trees.
- Leaves and nodes insert verification.
- Data types for tree objects.
- Scope of the tree objects.
- Tree classes implementation.
- Restrictions on trees when used as a SmartNavigation data source.
- Tree classes reference.

Understanding Tree Classes

Using the tree classes in your PeopleCode, you have access to all the functionality of PeopleSoft Tree Manager. Your application should instantiate the appropriate tree objects when it must work with the tree system database data, call the appropriate methods and properties, then close the objects when it is finished.

Creating or deleting a tree *object* does not create or delete tree system database information. You must call the method for that tree object directly to create or delete database information, that is, the Create or Delete method.

One instance of a tree or tree structure object can be created to work on multiple database entities. However, only one tree or tree structure can be open at a time. If you open a Tree before closing the one that's currently open, you receive an error. You must explicitly close a tree or tree structure (using the Close method) before you try to open a second one.

All of the classes, and most of the properties and methods that make up the Tree Classes have a GUI representation in PeopleSoft Tree Manager. This document assumes that the reader is familiar with PeopleSoft Tree Manager.

The following classes make up the tree classes:

- Branch Collection
- Leaf

- Level
- Level Collection
- Node
- Tree
- Tree Structure

With most of the classes of objects in PeopleTools, when you use a *Getxxx* method, you are fully instantiating an object. However, for the tree class, when you use GetTree (from the Session object), you get a closed tree. A closed tree is a tree object with just the key fields filled in. The rest of the data is not present. To open the tree, you must use the Open method. Working with closed trees can improve your performance. The same applies to a tree structure: it's closed when you get it from the Session object, and you must open it before you can access its properties or change it.

There aren't any built-in functions for the tree classes: objects are instantiated from identifiers, from other objects, or from a Session object.

Using the GenerateTree function, you can produce a GUI representation of a tree in the PeopleSoft Pure Internet Architecture.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TreeDetailInNode

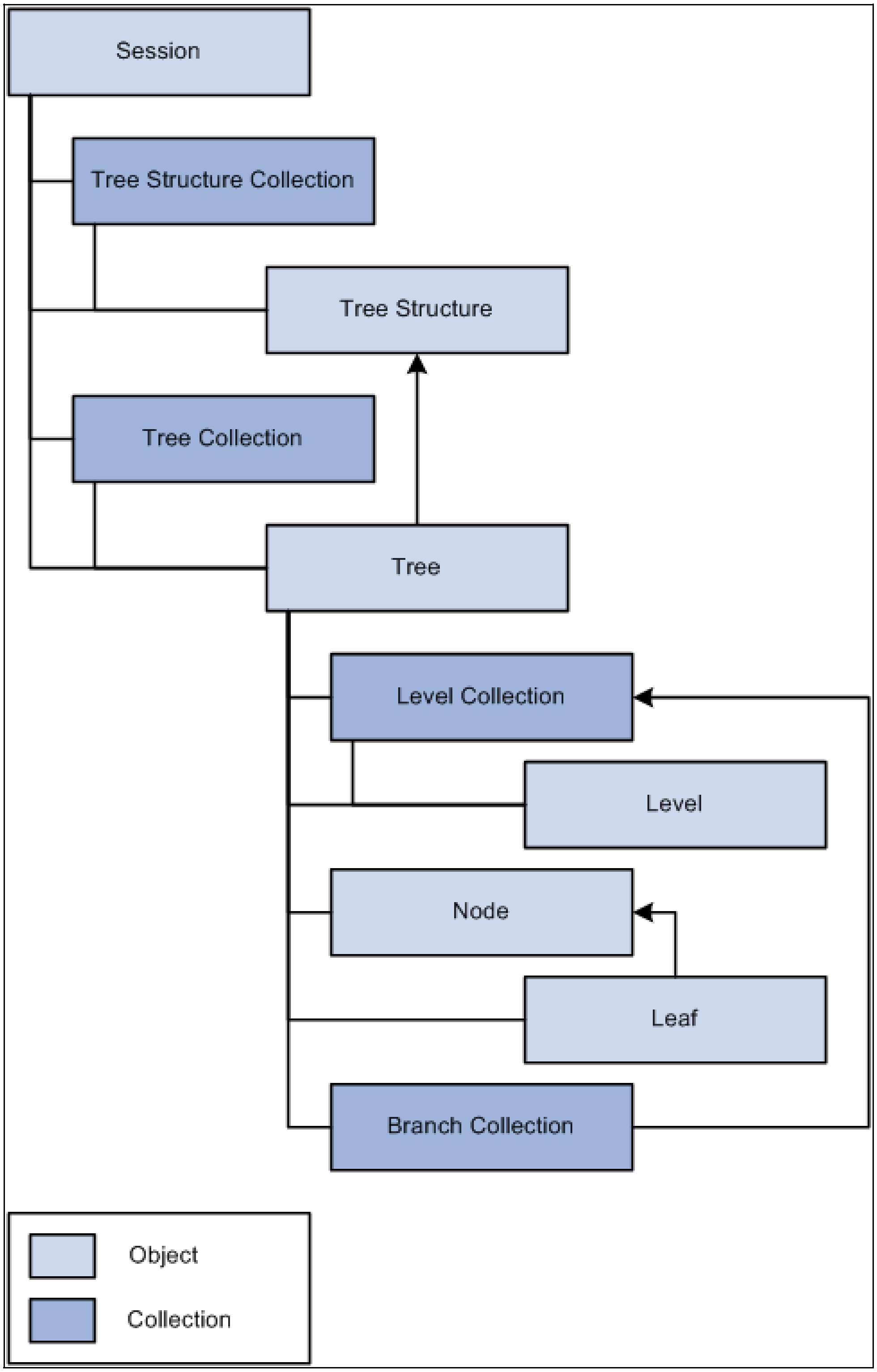
Chapter 38, "Session Class," page 2179

PeopleTools 8.51 PeopleBook: PeopleCode Developer's Guide, "Using HTML Trees and the GenerateTree Function," Using the GenerateTree Function

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Introduction to PeopleSoft Tree Manager"

Relationships Between Different Tree Classes

The following diagram shows the tree classes and their relationships. Objects at the tip of the arrowhead are accessed or created from the object at the left of the arrowhead. Squares indicate objects. Ovals indicate collections.



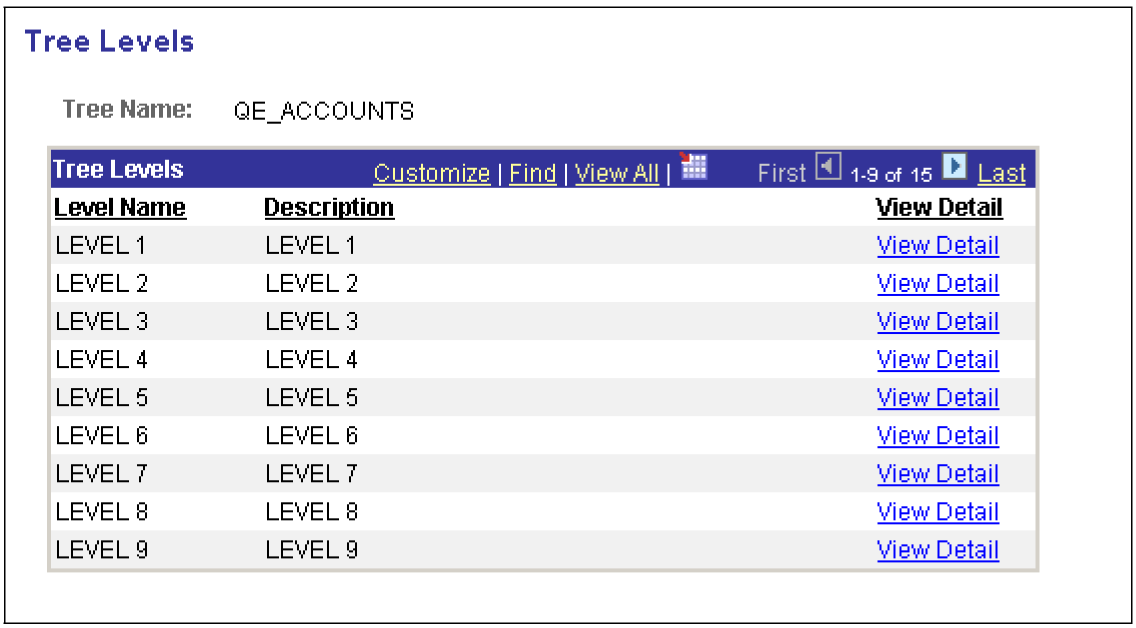
Tree classes and their relationships

- From the session object, you can get identifiers for a tree or a tree structure.
- From the level collection object, you can instantiate a level object.
- From the branch collection object, you can get an identifier for a branch.

- From the tree object, you can instantiate a tree, a tree structure, a branch collection, a leaf, a node, and a level collection.
- From the node object, you can instantiate a leaf object or a node object.
- From the leaf object, you can instantiate a leaf object or a node object.

Collections in the Tree Classes

A *collection* is a set of similar things, like a group of already existing branches or levels. Like everything else in the tree classes, collections have a GUI representation. For example, when you find an existing branch from PeopleSoft Tree Manager, you get a dialog that lets you select the branches you want. This dialog box data is represented in PeopleCode as the *tree level collection.*



Example of level collection

The following collections are part of the tree classes:

- Branch collection
- Level collection

Error Handling With Trees

The tree classes log descriptive information regarding errors and warnings to the PSMessages collection, instantiated from a session object.

In your program, use the PSMessages collection to identify and report to the user any errors that are encountered during processing.

The tree classes log errors "interactively", that is, as they happen. For example, suppose you had created a new tree, and were setting the effective date using the effective date property (EffDt). If you used an invalid value for the effective date, the error would be logged in the PSMessages collection as soon as you set the value, not when you saved the tree.

When you want to check for errors depends on your application. However, if you check for errors after every assignment, you may see a performance degradation.

One way to check for errors is to check the number of messages in the PSMessages collection, using the Count property. If the Count is 0, no error or warning messages have been logged to the PSMessages collection.

```
Local ApiObject &MYSESSION;
Local ApiObject &ERRORCOL;
Local ApiObject &TREE, TREELIST;

&MYSESSION = %Session;

If &MYSESSION Then
   /* connection is good */

&MyTree = &Session.GetTree();
    /* open a tree */
   &TreeReturn = &MyTree.Open("", "", "PERSONAL_DATA", "1999-06-01", "",true);
    /* Do error checking */

   &ERRORCOL = &MYSESSION.PSMessages;
   If (&ERRORCOL.Count <> 0) Then
      /* errors occurred - do processing */
   Else
      /* no errors */
   End-If;
   Else
   /* do processing for no connection */
End-If;
```

You can also do an error check based on the return value of your API calls

```
If ALL(&TreeReturn) then
   /* processing errors */
End-if:
```

See Also

Chapter 38, "Session Class," Error Handling, page 2180

Leaves and Nodes Insert Verification

InsertChildLeaf, InsertSibNode, InsertChildNode, and so on, can return False or a Null object reference, depending upon the type of error encountered. You may want to declare references to new leaves or nodes as type ANY until after you verify they were actually created. You can do this by using the None or All PeopleCode functions to determine whether a valid Leaf or Node Object was created. If a Leaf or Node object was created ALL()returns True and None() returns False.


```
&NewLeaf = &RootNode.InsertChildLeaf("8000", "8999");
If NONE(&NewLeaf) Then
   /* Leaf not inserted, do error processing */
   &Messages = &Session.PSMessages;
   If &Messages.WarningPending Then
      /* do error processing */
   End-if;
End-if;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," None

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All

Restrictions on Trees When Used as a SmartNavigation Data Source

When a tree is used as a SmartNavigation data source, the values of several tree-specific properties are passed to the SmartNavigation application via URL. Certain characters are inappropriate for use in a URL and must be avoided. When using a tree as a SmartNavigation data source, do not use any of the following characters in the tree name, setID, user key value, and tree branch values:

- pound (#)
- percent (%)
- dollar ($)
- ampersand (&)
- plus (+)
- comma (,)
- forward slash/virgule (/)
- colon (:)
- semi-colon (;)
- equals (=)
- question mark (?)
- at symbol (@)
- space ( )
- quotation marks(")
- less than symbol (<)
- greater than symbol (>)
- left curly brace ({)
- right curly brace (})
- vertical bar/pipe (|)
- backslash (\)
- caret (^)
- tilde (~)
- left square bracket ([)
- right square bracket (])
- grave accent (`)

When using Tree class methods such as Copy, Create, Rename, SaveAs, SaveAsDraft, and others, take care to avoid using these invalid characters in the values for the tree name, setID, user key value, and tree branch.

See Also

PeopleTools 8.51 PeopleBook: PeopleTools Portal Technologies, "Administering Portals," Defining SmartNavigation Folders

Data Types for Tree Objects

All tree objects, that is, trees, tree structures, nodes, levels, and so on, are declared as type ApiObject. For example:

```
Local ApiObject &MYTREE;
```

```
Global ApiObject &MYNODE;
```

All tree objects can be declared as Local, Component or Global.

Scope of the Tree Objects

All tree objects, that is, trees, tree structures, nodes, leaves, and so on, can *only* be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

You can instantiate a tree or tree structure object only from a session object. You have to instantiate the session object, and connect to the database, before you can instantiate a tree or tree structure.

```
Local ApiObject &TREELIST;
Local ApiObject &MYSESSION;

&MYSESSION = %Session;

If &MYSESSION Then
   /* connection is good */
Else
   /* do error processing */
End-if;
```

Note. Tree Classes are accessible only through PeopleCode.

Tree Classes Implementation

You will often want to create a new tree. The following procedure discusses this action in more detail.

The TreeMover Application Engine program uses the Tree API (and File Layouts) for importing a tree from a flat file or exporting a tree to a flat file.

To create a new tree:

In this example, you are creating a new tree based on an existing tree structure. The following is the complete code sample: the steps explain each line.


```
Local ApiObject &Session;
Local ApiObject &TreeList, &MyTree;
Local ApiObject &LvlColl;

&Session = %Session;

&MyTree = &Session.GetTree();
   /* create new tree */

If All(&MyTree) Then
  &TreeReturn = &MyTree.Create("", "", "PERSONAL_DATA2", "1999-06-01", "PERSONAL_⇒
DATA");

   If &TreeReturn <> 0 then
      /* check PSMessages collection */
   End-if;

   &MyTree.description = "test tree";

   /* add level */
   &LvlColl = &MyTree.levels;
   &Level = &LvlColl.add("FIRST LVL");
   &Level.description = "First Level";

   /* add root node */
   &RootNode= &MyTree.insertroot("00001");

If ALL(&RootNode) Then
 /* insert a leaf */
       &NewLeaf = &RootNode.InsertChildLeaf("8000", "8999");

   /* save new tree */
   &RSLT = &MyTree.Save();

   /* Do error checking */
   If &RSLT <> 0 Then
      /* errors occurred = do error checking */
      &ERRORCOL = &Session.PSMessages;
      For &I = 1 To &ERRORCOL.count
         /* do error processing */
      End-For;
   Else
      /* no errors - saved correctly - do other processing */
   End-If;
End-if;

End-if;
```

1. Get a session object.

   Before you can get a tree, you have to get a session object. The session controls access to the tree, provides error tracing, enables you to set the runtime environment, and so on. Use the %Session system variable to return a reference to the current PeopleSoft session.

   ```
   &Session = %Session;
   ```

2. Get a tree object.

   Use the GetTree method specifying a null value `(" ")` to return a closed tree object.

   ```
   &MyTree = &Session.GetTree();
   ```


3. Create the Tree.

   The Create method creates a new tree with the name PERSONAL_DATA2. To ensure that you have a valid tree, use the All built-in function. Description is a required property (if you don't specify something for Description you cannot save the tree.)

   ```
   &TreeReturn = &MyTree.Create("", "", "PERSONAL_DATA2", "1999-06-01", "PERSONAL_⇒
   DATA");
   ```

   ```
   &MyTree.description = "test tree";
   ```

4. Add a level.

   To add a level, you have to instantiate a level collection. Although there aren't any levels in the tree, you can still access this collection. Use the Add method with the level collection to add a new level. Remember, the level name must be 8 characters or less. Description is a required property (if you don't specify something for Description you cannot save the tree.)

   ```
   &LvlColl = &MyTree.levels;
   &Level = &LvlColl.add("FIRST LVL");
   &Level.description = "First Level";
   ```

5. Add the root node.

   Because this is a new tree, you must first add the root node.

   ```
   &RootNode = &MyTree.insertroot("00001");
   ```

6. Add a leaf.

   To add a new leaf, you must have a reference to the parent node object. Using the All built-in function ensures that there is a root node before you try to insert the leaf with the InsertChildLeaf method.

   ```
   If ALL(&RootNode) Then
      &NewLeaf = &RootNode.InsertChildLeaf("8000", "8999");
   ```

7. Save the tree.

   When you execute the Save method, the new tree is saved to the database.

   ```
   &RSLT = &MyTree.Save();
   ```

   Note. If you're running the tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

   ---

8. Check for errors.

   You can check if there were any errors using the PSMessages property on the session object.

```
If All (&RSLT) Then
   /* errors occurred = do error checking */
   &ERRORCOL = &Session.PSMessages;
   For &I = 1 To &ERRORCOL.count
      /* do error processing */
   End-For;
Else
   /* no errors - saved correctly - do other processing */
End-If;
```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error.

Note. If you've called the Tree API from an Application Engine program, all errors are also logged in the application engine error log tables.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Using TreeMover"

Tree Classes Reference

This section provides a reference to the following topics:

- Session class methods in the tree API.
- Branch collection.
- Leaf class.
- Level collection.
- Level class.
- Node class.
- Tree class.
- Tree structure class.

Session Class Methods

Tree Classes don't have any built-in functions. Instead, they're instantiated from the Session Class.

In this section, we discuss the Session class methods. The methods are discussed in alphabetical order.

GetTree

Syntax

GetTree()

Description

The GetTree method returns a closed tree object. Before you can use some of the methods or any of the properties, you must use the Open method to open a tree object.

Parameters

None.

Returns

A reference to a closed tree object.

Example

To create a new tree, use the following:

```
&MYTREE = %Session.GetTree();
&MYTREE.Create("","","PERSONAL_NEW", "05-05-1997", "PERSONAL_DATA");
```

To determine if a tree already exists in the database, use the following code:

```
&MYTREE = %Session.GetTree();
If &MYTREE.Exists("","","PERSONAL_OLD", "05-05-1997", "PERSONAL_DATA") = 0 then
     /* Do processing */
End-If;
```

To open an existing tree, use the following code:

```
&MYTREE = %Session.GetTree();
&MYTREE.Open("","","PERSONAL_OLD", "05-05-1997", "PERSONAL_DATA", True);
```

GetTreeStructure

Syntax

GetTreeStructure()

Description

The GetTreeStructure method returns a closed tree structure object. Before you can use some of the methods or any of the properties, you must use the Open method to open the tree structure object.

Branch Collection

A branch collection is returned from the Branches property, used with an open tree object.

```
&MYBRANCHCOLL = &MYTREE.Branches;
```

&MYTREE must be a branched tree. To verify whether a tree is branched, you can check the value of the IsBranched property of a tree object (Boolean value: True or False.)

See Chapter 43, "Tree Classes," Branches, page 2386.

Branch Collection Property

In this section, we discuss the Branch collection properties. The properties are discussed in alphabetical order.

Count

Description

Returns the number of branches for the branch collection object.

Leaf Class

Leaf objects are instantiated from other tree classes as follows:

- From a tree object with the FindLeaf method.
- From a node object with the FirstChildLeaf property.
- From another leaf object with the NextSib and PrevSib properties.

See Chapter 43, "Tree Classes," FindLeaf, page 2364; Chapter 43, "Tree Classes," FirstChildLeaf, page 2348; Chapter 43, "Tree Classes," NextSib, page 2317 and Chapter 43, "Tree Classes," PrevSib, page 2318.

Leaf Class Methods

In this section, we discuss the Leaf class methods. The methods are discussed in alphabetical order.

Cut

Syntax

Cut()

Description

The Cut method cuts the leaf and puts it into a buffer (the clipboard.) You can then use the PasteSib method to add the leaf back as a sibling of a different leaf. You can also use the PasteChild node method to add the leaf back as a child of a different node.

You can only have one object at a time on the clipboard. If you cut another leaf or node, the leaf in the clipboard is overwritten.

Parameters

None.

Returns

A number; 0 if the cut is successful.

See Also

Chapter 43, "Tree Classes," Cut, page 2327; Chapter 43, "Tree Classes," PasteSib, page 2312 and Chapter 43, "Tree Classes," PasteChild, page 2342

Delete

Syntax

Delete()

Description

The Delete method deletes the leaf object executing the method *from the database.*

Returns

A number; 0 if the delete is successful.

Example

```
&MYLEAF = &MYTREE.FindLeaf("8200", "8300");
&MYLEAF.Delete();
```

DeleteByRange

Syntax

DeleteByRange(*RangeFrom*,*RangeTo*)

Description

The DeleteByRange method deletes the specified leaf object from the database. The leaf specified by the parameters does *not* have to match the leaf executing the method. That is, if &MYLEAF is associated with a range 8200-8300, you can specify a different range with the DeleteByRange method. For example:

```
&MYLEAF = &MYTREE.FindLeaf("8200", "8300");
/* some processing */
&RET_VALUE = &MYLEAF.DeleteByRange("8400", "8500");
```

Parameters

Parameter	***Description***
RangeFrom	Specify the starting range of the leaf to be deleted. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf to be deleted. This parameter takes a string value.

Returns

A number; 0 if the delete is successful.

Example

```
&RET_VALUE= &MYLEAF.DeleteByRange("8400", "8500");
```

GenABNMenuElement

Syntax

`GenABNMenuElement``()`

Description

Use this method to generate a single <li> element for this tree leaf.

The <li> element from this leaf and the <li> elements from its sibling leaves and nodes are concatenated to form an HTML code fragment to be consumed by the portal.

Parameters

None.

Returns

A string representing the <li> element for this tree leaf.

Example

```
&LI_list = &LI_list | &ChildLeaf.GenABNMenuElement();
```

InsertDynSib

Syntax

`InsertDynSib``()`

Description

The InsertDynSib method inserts a dynamic leaf as a sibling leaf to the leaf object executing the method. The leaf is a *new*node.

A leaf object associated with the new leaf is returned. The new leaf is inserted as the next sibling leaf. If the new leaf isn't inserted successfully, the method returns Null.

Note. To insert a sibling leaf with a specific range, use the InsertSib method.

Parameters

None.

Returns

A reference to the new leaf. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWLEAF = &MYLEAF.InsertDynSib();
```

See Also

Chapter 43, "Tree Classes," InsertSib, page 2306

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None

InsertSib

Syntax

`InsertSib`*`(RangeFrom,RangeTo)`*

Description

The InsertSib method inserts a new leaf as a sibling leaf under the leaf currently executing the method. The leaf specified by the range parameters must be a *new* leaf. You receive an error if the leaf already exists.

A leaf object associated with the new leaf is returned. The new leaf is inserted as the next sibling leaf, although there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields.

Note. To insert a dynamic sibling leaf (that is, without specifying a range) use the InsertDynSib method.

Parameters

Parameter	*Description*
RangeFrom	Specify the starting range of the new leaf. This parameter takes a string value.

Parameter	Description
RangeTo	Specify the ending range of the new leaf. This parameter takes a string value.

Returns

A leaf object associated with the new leaf. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWLEAF = &MYLEAF.InsertSib("10090", "10100");
```

See Also

Chapter 43, "Tree Classes," InsertDynSib, page 2305

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None

LoadABNChart

Syntax

LoadABNChart*(&chart_rowset,&relactions_rowset*)

Description

Use this method to load a child leaf of the current tree node into the SmartNavigation chart rowset and the component buffer.

Parameters

Parameter	Description
&chart_rowset	Specifies the SmartNavigation chart rowset. Typically, this is the rowset returned by the GetABNChartRowSet function.
&relactions_rowset	Specifies the related actions rowset. Typically, this is the rowset returned by the GetABNRelActnRowSet function.

Returns

None.

Example

```
&ChildLeaf.LoadABNChart(&chart_rs, &ra_rs);
```

See Also

Chapter 43, "Tree Classes," LoadABNChartOrdered, page 2308

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetABNChartRowSet and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetABNRelActnRowSet

LoadABNChartOrdered

Syntax

LoadABNChartOrdered*(&chart_rowset,&relactions_rowset, order)*

Description

Use this method to load a child leaf of the current tree node into the SmartNavigation chart rowset and the component buffer. The order for the leaf within its chart level is specified by this method.

Parameters

Parameter	***Description***
&chart_rowset	Specifies the SmartNavigation chart rowset. Typically, this is the rowset returned by the GetABNChartRowSet function.
&relactions_rowset	Specifies the related actions rowset. Typically, this is the rowset returned by the GetABNRelActnRowSet function.
order	Specify the order for the leaf as a number. **Note.** For a given chart level, all chart leaves must have a display order greater than zero.

Returns

None.

Example

```
&ChildLeaf.LoadABNChartOrdered(&chart_rs, &ra_rs, 5);
```

See Also

Chapter 43, "Tree Classes," LoadABNChart, page 2307

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetABNChartRowSet and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetABNRelActnRowSet

MoveAsChild

Syntax

`MoveAsChild`*`(Node)`*

Description

The MoveAsChild method moves the leaf executing the method to a different node in the tree. The leaf becomes the first child leaf under the node, even though there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields. The specified node becomes the new parent to the leaf.

Parameters

Parameter	***Description***
Node	Specify a node object. This parameter value must be an object, not a string or a name.

Note. To specify a node name, not a node object, use the MoveAsChildByName method.

Returns

A number; 0 if the move is successful.

Example

The following example moves leaf 8001 from node 10100 (old parent) to node 00001 (new parent)

```
&MY_LEAF = &MY_TREE.FindLeaf("8001", "8001");
&NEW_PARENT = &MY_TREE.FindNode("00001", "");
If &NEW_PARENT <> Null Then
   &RET_VALUE = &MY_LEAF.MoveAsChild(&NEW_PARENT);
End-If;
```

See Also

Chapter 43, "Tree Classes," MoveAsChildByName, page 2310

MoveAsChildByName

Syntax

MoveAsChildByName*(NodeName)*

Description

The MoveAsChildByName method moves the leaf executing the method to a different node in the tree. The leaf becomes the first child leaf under the node, although there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields. The specified node becomes the new parent to the leaf. The node specified by *NodeName* must be a valid node name.

Parameters

Parameter	***Description***
NodeName	Specify the name of a node. *NodeName* must be a valid node for the existing tree. This parameter takes a string value.

Note. To specify a node object, not a node name, use the MoveAsChild method.

Returns

A number; 0 if the move is successful.

Example

The following example moves leaf 8001 from node 10100 (old parent) to node 00001 (new parent)


```
&MY_LEAF = &MY_TREE.FindLeaf("8001", "8001");
If &MY_LEAF <> Null Then
   &RET_VALUE = &MY_LEAF.MoveAsChildByName("00001");
End-If;
```

See Also

Chapter 43, "Tree Classes," MoveAsChild, page 2309

MoveAsSib

Syntax

MoveAsSib*(Leaf)*

Description

The MoveAsSib method moves the leaf executing the method to a new place in the tree. The current leaf becomes the next sibling leaf under the specified leaf object, although there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields.

Parameters

Parameter	*Description*
Leaf	Specify a leaf object. This parameter value must be an object, not a string or a name.

Note. To specify a range, not a leaf object, use the MoveAsSibByRange method.

Returns

A number; 0 if the move is successful.

Example

```
&MYLEAF = &MYTREE.FindLeaf("8000", "8000");
&MYLEAF2 = &MYTREE.FindLeaf("9000", "9000");
&RET_VALUE = &MYLEAF.MoveAsSib(&MYLEAF2);
```

See Also

Chapter 43, "Tree Classes," MoveAsSibByRange, page 2312

MoveAsSibByRange

Syntax

`MoveAsSibByRange``(`*`RangeFrom`*`,`*`RangeTo`*`)`

Description

The MoveAsSibByRange method moves the leaf executing the method to a new place in the tree. The current leaf becomes the next sibling leaf under the specified leaf object.

Parameters

Parameter	***Description***
RangeFrom	Specify the starting range of the leaf that you want as the parent of the leaf executing the method. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf that you want as the parent of the leaf executing the method. This parameter takes a string value.

Note. To specify a leaf object, not a range, use the MoveAsSib method.

Returns

A number; 0 if the move is successful.

Example

```
&MYLEAF = &MYTREE.FindLeaf("8000", "8000");
&RET_VALUE = &MYLEAF.MoveAsSibByRange("9000", "9000");
```

See Also

Chapter 43, "Tree Classes," MoveAsSib, page 2311

PasteSib

Syntax

`PasteSib``()`

Description

The PasteSib method makes the leaf from the buffer (clipboard) a sibling to the leaf executing the method. You must use the Cut leaf method before you can paste a leaf.

You can have only one object at a time on the clipboard. If you cut another leaf or node, the leaf in the clipboard is overwritten.

Parameters

None.

Returns

A number; 0 if the paste is successful.

See Also

Chapter 43, "Tree Classes," Cut, page 2303; Chapter 43, "Tree Classes," PasteChild, page 2342 and Chapter 43, "Tree Classes," Cut, page 2327 node method

RefreshDescription

Syntax

`RefreshDescription`*`(expand_range)`*

Description

Use this method to reload the tree leaf's description from the database.

Note. This method is currently supported for what is known as a single detail leaf. Do not use this method with ranged or dynamic leaves.

Parameters

Parameter	***Description***
expand_range	Specify a Boolean value; however, this parameter is currently ignored. **Note.** This parameter is reserved for future use.

Returns

A number: 0 if the refresh was successful.

See Also

Chapter 43, "Tree Classes," Description, page 2315

UpdateRanges

Syntax

`UpdateRanges`*`(RangeFrom,RangeTo)`*

Description

The UpdateRanges method performs validation of edited ranges, updating as necessary.

Parameters

Parameter	***Description***
RangeFrom	Specify the starting range of the leaf that you updated. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf you want updated. This parameter takes a string value.

Returns

A number; 0 if the update is successful.

Leaf Class Properties

In this section, we discuss the Leaf class properties. The properties are discussed in alphabetical order.

Description

Description

Use this property to return the tree leaf's description as a string.

Note. This property is currently supported for what is known as a single detail leaf. Do not use this property with ranged or dynamic leaves.

This property is read-only.

See Also

Chapter 43, "Tree Classes," RefreshDescription, page 2313

DisplayLevelNumber

Description

When using the Tree API to create a graphic representation of the tree (such as for HTML Tree Manager or other application pages) use this property to indicate the *display* level, that is, tell the user how many levels deep the leaf is. This is generally used for trees where levels aren't used and the LevelNumber property always returns 0. This property always returns a number.

This property is read-only.

See Also

Chapter 43, "Tree Classes," LevelNumber, page 2352

Dynamic

Description

This property specifies whether the leaf has a dynamic range or a specified range. If you set this property to True, the leaf has a dynamic range. If this is a new leaf, and you do not set this property, the value is automatically set is False.

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Creating Trees"

HasNextSib

Description

This property returns True if the leaf has a next sibling, that is, it isn't the last leaf listed under the parent node.

This property is read-only.

HasPrevSib

Description

This property returns True if the leaf has a previous sibling, that is, it isn't the first leaf listed under the parent node.

This property is read-only.

ImageName

Description

This property sets the which image is used to display the leaf. This property takes a string value for the image name of the leaf.

This property is read-write.

IsChanged

Description

This property returns True if the leaf has been edited or changed.

This property is read-only.

Example

```
If &MYLEAF.IsChanged Then
   &MYTREE.Save();
End-If;
```

IsCut

Description

This property returns True if the leaf has been cut from the displayed tree. This property is generally used with the HTML Tree Manager.

This property is read-only.

IsDeleted

Description

This property returns True if the leaf has been deleted from the tree but the tree hasn't been saved.

This property is read-only.

IsInserted

Description

This property returns True if the leaf has been inserted as a new leaf in the tree but the tree hasn't been saved.

This property is read-only.

NextSib

Description

This property returns a leaf object associated with the next sibling leaf. The next sibling of a leaf is the leaf appearing under the current leaf. If there is no next sibling and you try to assign it to a variable, you receive a runtime error.

This property is read-only.

Example

The following code traversed the leaves from top to bottom.

```
While &MYLEAF.HasNextSib
   &MYLEAF = &MYLEAF.NextSib;
   /* do some processing */
End-While;
```

Parent

Description

This property returns a node object associated with the parent node for the leaf.

This property is read-only.

Example

```
&PARENTNODE = &MYLEAF.Parent;
/* do node processing with node object */
```

PrevSib

Description

This property returns a leaf object associated with the previous sibling leaf. The previous sibling of a leaf is the leaf appearing above the current leaf. If there is no previous sibling and you try to assign it to a variable, you receive a runtime error.

This property is read-only.

Example

The following code traverses the leaves from bottom to top.

```
While &MYLEAF.HasPrevSib
   &MYLEAF = &MYLEAF.PrevSib;
   /* do some processing */
End-While;
```

RangeFrom

Description

This property returns the starting range, as a string, of the leaf.

This property is read-write.

RangeTo

Description

This property returns the ending range, as a string, of the leaf.

This property is read-write.

TreeBranchName

Description

This property returns the branch name of the tree as a string if the tree is branched. If not branched, this property returns a blank string.

This property is read-only.

TreeEffDt

Description

This property returns the effective date of the tree as a string if the tree is effective-dated. If not effective-dated, this property returns a blank string.

This property is read-only.

TreeName

Description

This property returns the name of the tree as a string.

This property is read-only.

TreeSetId

Description

This property returns the SetID of the tree as a string if the tree has a SetID. If the tree doesn't have a SetID, this property returns a blank string.

This property is read-only.

TreeUserKeyValue

Description

This property returns the UserKeyValue of the tree as a string if the tree has a UserKeyValue. If the tree doesn't have a UserKeyValue, this property returns a blank string.

This property is read-only.

Level Collection

Level collection is instantiated from a tree object with the Levels property.

See Chapter 43, "Tree Classes," Levels, page 2394.

Level Collection Methods

In this section, we discuss the Level Collection methods. The methods are discussed in alphabetical order.

Add

Syntax

`Add`*`(LevelName)`*

Description

The Add method adds a new level called *LevelName* to the database. The specified level must be a new level, that is, it cannot exist in the database. *LevelName* takes a string value.

Note. *LevelName* must be 8 characters or less.

If no levels currently exist in the tree, the level is added at the top level. If levels already exist in the tree, the new level is added as the last level. You can change the level number of a level using the Number property.

If the new level is the first level, the AllValuesAudit property is automatically set to True.

The new level is notadded to the database until the tree is explicitly saved.

This method returns a reference to the new level object.

See Also

Chapter 43, "Tree Classes," Number, page 2324 and Chapter 43, "Tree Classes," AllValuesAudit, page 2324

Item

Syntax

Item*(LevelName,LevelNumber)*

Description

The Item method returns a reference to the specified level in the level collection executing the method as an object. The *LevelName* parameter specifies the name of the level to access. This parameter takes a string value. The *LevelNumber* parameter specifies the number at which the level exists. This parameter takes a number value. For example, suppose your level collection contains the following levels, in this order:

1. CORPORATE
2. COMPANY
3. DIVISION
4. DEPARTMENT
5. BRANCH

You want to access the fourth level, DEPARTMENT. You would use the following code:

```
&MYLEVEL = &LVLCOLLECTION.Item("DEPARTMENT", 4);
```

Remove

Syntax

Remove()

Description

The Remove method deletes the current level from the database. You can use this method only after you have used the First, Next, or Item properties: otherwise the system doesn't know which level to delete from the tree. If no level has been indicated yet system tries to remove the last level. If the level to remove has nodes associated with it, the system doesn't remove the level. The rest of the levels in the collection after the deleted level are moved up in the list and renumbered so that the levels remain consecutively numbered.

Returns

A number; 0 if the level is successfully removed.

Level Collection Properties

In this section, we discuss the Level collection properties. The properties are discussed in alphabetical order.

Count

Description

Returns the number of levels for the level collection object.

First

Description

The First property returns a reference to the first level in the level collection executing the method as an open object.

Last

Description

The Last property returns a reference to the last level in the level collection executing the method as an open object.

Next

Description

The Next property returns a reference to the next level in the level collection executing the method as an open object. You can use this method only after you have used either the First or Item properties: otherwise the system doesn't know where to start in the tree.

Level Class

Level objects are instantiated from the level class collection object with the Item method or one of the following properties:

- First
- Last
- Next

See Chapter 43, "Tree Classes," Item, page 2321; Chapter 43, "Tree Classes," First, page 2322; Chapter 43, "Tree Classes," Last, page 2322 and Chapter 43, "Tree Classes," Next, page 2322.

Level Class Methods

In this section, we discuss the Leaf class properties. The properties are discussed in alphabetical order.

Create

Syntax

`Create`*`(LevelName,LevelNumber)`*

Description

Note. This method has been deprecated. If you create a level using this method, there is no way of saving the level to the database. Use the Add level collection method instead.

Level Class Properties

In this section, we discuss the Level class properties. The properties are discussed in alphabetical order.

AllValuesAudit

Description

This property specifies whether PeopleSoft Tree Manager permits nodes to skip over this level. To allow nodes to skip this level, specify this parameter as True. If you are creating a new level, and this level is the first level in a tree, this property is automatically set to True. If the level isn't the first level, this property is set to False by default..

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Using PeopleSoft Tree Manager," Modifying Tree Definitions

Description

Description

This property returns the description of the level.

This property is read-write.

Name

Description

This property returns the name of the level.

This property is read-write.

Number

Description

This property returns the number of the level.

Note. Though you can use this property to set the level of the number, you should not do so, as level numbers are automatically assigned when the level is inserted to the level collection using the Add method.

This property is read-write.

TreeBranchName

Description

This property returns the branch name of the tree as a string if the tree is branched. If not branched, this property returns a blank string.

This property is read-only.

TreeEffDt

Description

This property returns the effective date of the tree as a string if the tree is effective-dated. If not-effective dated, this property returns a blank string.

This property is read-only.

TreeName

Description

This property returns the name of the tree as a string.

This property is read-only.

TreeSetId

Description

This property returns the SetID of the tree as a string if the tree has a SetID. If the tree doesn't have a SetID, this property returns a blank string.

This property is read-only.

TreeUserKeyValue

Description

This property returns the UserKeyValue of the tree as a string if the tree has a UserKeyValue. If the tree doesn't have a UserKeyValue, this property returns a blank string.

Node Class

Node objects are instantiated from other tree classes, as follows:

- From a tree object with the FindNode method.
- From a leaf object with the Parent property.
- From another node object with the FirstChildNode, NextSib, PrevSib, and Parent properties.

See Chapter 43, "Tree Classes," FindNode, page 2365; Chapter 43, "Tree Classes," FirstChildNode, page 2348; Chapter 43, "Tree Classes," NextSib, page 2353 and Chapter 43, "Tree Classes," PrevSib, page 2353.

See Chapter 43, "Tree Classes," Parent, page 2318.

See Chapter 43, "Tree Classes," Parent, page 2353.

Node Class Methods

In this section, we discuss the Node class methods. The methods are discussed in alphabetical order.

Branch

Syntax

Branch()

Description

The Branch method branches the node executing this method, identifying all of its child nodes and leaves as part of that branch. *Branching* means taking a limb of a tree and creating another subtree to hold that limb. (Technically is it not creating a real tree.) The subtree is accessed through the Branches collection.

After you use the Branch method, you can no longer access the child nodes and leaves of the node that executed the method until you close the current tree, open the new branched tree, and find the node again. To unbranch the tree, use the Unbranch method.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("10900", "");
&MYNODE.Branch();
```


See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Creating Trees," Working with Tree Branches

Cut

Syntax

Cut()

Description

The Cut method cuts the node executing the method and puts it into a buffer (the clipboard). You can then use the PasteSib method to add the node back as a sibling of a different node. You can also use the PasteChild node method to add the node back as a child of a different node.

You can have only one object at a time on the clipboard. If you cut another leaf or node, the node on the clipboard is overwritten.

Parameters

None.

Returns

A number; 0 if method is successful.

See Also

Chapter 43, "Tree Classes," PasteSib, page 2343; Chapter 43, "Tree Classes," PasteChild, page 2342 and Chapter 43, "Tree Classes," Cut, page 2303 leaf method

Delete

Syntax

Delete()

Description

The Delete method deletes the node object executing the method *from the database.*

Returns

A number; 0 if method is successful.

DeleteByName

Syntax

`DeleteByName`*`(NodeName)`*

Description

The DeleteByName method deletes the specified node from the database. The *NodeName* parameter takes a string value. The node specified by *NodeName* does not have to match the node executing the method. That is, if &MYNODE is associated with node 100200, you can specify a different node with the DeleteByName method. For example:

```
&MYNODE = &MYTREE.FindNode("100200", "");
/* some processing */
&MYNODE.DeleteByName("100300");
```

The node name specified by *NodeName* must be a valid node in the existing open tree object. If *NodeName* doesn't exist, you receive a runtime error.

Returns

A number; 0 if method is successful.

Expand

Syntax

`Expand`*`(ExpandType)`*

Description

The Expand method gets the next level of nodes or leaves into memory from the selected node. What leaves or nodes get expanded depends on the *ExpandType*.

Parameters

Parameter	***Description***
ExpandType	Specifies which leaves or nodes get expanded. Values are:

Value	*Description*
0	Expand only nodes
1	Expand nodes and leaves
2	Expand only one level

Returns

A number; 0 if method is successful.

Example

```
If (&MYNODE.State = 2 AND &MYNODE.HasChildren);
   /* if node is collapsed */
   &MYNODE.Expand(2);
End-if;
```

GenABNMenuElement

Syntax

GenABNMenuElement(*initial_node*)

Description

Use this method to generate a single <li> element for this tree node.

The <li> element from this node and any sibling nodes and leaves are concatenated to form an HTML code fragment to be consumed by the portal.

Parameters

Parameter	*Description*
initial_node	Specifies the initial chart node. Typically, this is returned directly by calling the GetABNInitialNode function.

Returns

A string representing the <li> element for this tree node.

Example

```
&LI_list = &LI_list | &ChildNode.GenABNMenuElement(GetABNInitialNode(&reqParams));
```

GenBreadCrumbs

Syntax

GenBreadCrumbs*(list)*

Description

Call the GenBreadCrumbs method from the requested node to generate an HTML code fragment that will be rendered in the browser as breadcrumbs.

The <li> elements in the input string are created by the GenRelatedActions and GenABNMenuElement methods of the Node class, and the GenABNMenuElement method of the Leaf class.

Parameters

Parameter	*Description*
list	Specifies the list of elements as a string.

Returns

A string with the <li> elements for the bread crumbs for this tree node appended.

Example

The following example shows how the list of <li> elements is created for a node. In this example, <li> elements are generated for related actions, the first child node, and child leaves. This list is then passed to the GenHTMLMenu function.

```
&LI_lis = &MyNode.GenBreadCrumbs(&LI_list);
```

See Also

Chapter 43, "Tree Classes," GenABNMenuElement, page 2305

Chapter 43, "Tree Classes," GenABNMenuElement, page 2329 and Chapter 43, "Tree Classes," GenRelatedActions, page 2331

GenRelatedActions

Syntax

GenRelatedActions()

Description

Use this method to generate the list of <li> elements for the related actions menu for the current tree node. Call this function before calling GenABNMenuElement for the same node. The <li> elements from this function, from this node, from any child nodes and leaves traversed, and from the parent tree nodes are concatenated to form an HTML code fragment to be consumed by the portal.

Parameters

None.

Returns

A string representing the <li> elements for the related actions for this tree node.

Example

```
&LI_list = &LI_list | &MyNode.GenRelatedActions();
```

See Also

Chapter 43, "Tree Classes," GenABNMenuElement, page 2329 and Chapter 43, "Tree Classes," GenBreadCrumbs, page 2330

InsertChildLeaf

Syntax

InsertChildLeaf*(RangeFrom,RangeTo)*

Description

The InsertChildLeaf method inserts a new leaf (as specified by the range parameters) under the node object executing the method.

A leaf object associated with the new leaf is returned.

The new leaf is inserted as the child of the current node, although there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields.

Note. You must specify a range with this method. To insert a dynamic range leaf (that is, one with no range) use the InsertDynChildLeaf method.

Parameters

Parameter	***Description***
RangeFrom	Specify the starting range of the new leaf. This parameter takes a string value.
RangeTo	Specify the ending range of the new leaf. This parameter takes a string value.

Returns

A leaf object associated with the new leaf. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWLEAF = &MYLNODE.InsertChildLeaf("10090", "10100");

If None(&NEWLEAF) Then
   /* Error Processing */
End-If;
```

See Also

Chapter 43, "Tree Classes," InsertDynChildLeaf, page 2334

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None

InsertChildNode

Syntax

`InsertChildNode`*`(NodeName)`*

Description

The InsertChildNode method inserts a new node (as specified by *NodeName)* as a child node under the node object executing the method. The node specified by *NodeName* must be a new node. You receive an error if the node already exists.

A node object associated with the new node is returned. If the new node isn't inserted successfully, the method returns False or Null.

Parameters

Parameter	***Description***
NodeName	Specify a name for the new node. This parameter takes a string value. *NodeName* must not already exist.

Returns

A reference to the new node. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWNODE = &MYNODE.InsertChildNode("100500");

If None(&NEWNODE) Then
   /* Do error processing */
End-If;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and
PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," None

InsertChildRecord

Syntax

InsertChildRecord*(NodeName)*

Description

The InsertChildRecord method inserts a new record (as specified by *NodeName)* as a node under the parent node executing this method. This method works only on trees that are Query Access Trees.

A node object associated with the new node is returned, otherwise this method returns False or Null.

Parameters

Parameter	*Description*
NodeName	Specify an existing record name. This parameter takes a string value. This record must not already be a node under the parent node (that is, a parent node can't have the same record as a child node more than once.)

Returns

A reference to the new node. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWNODE = &MYNODE.InsertChildRecord("PERSONAL_DATA");

If None(&NEWNODE) Then
   /* Do error processing */
End-if;
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None

InsertDynChildLeaf

Syntax

```
InsertDynChildLeaf()
```

Description

The InsertDynChildLeaf method inserts a dynamic leaf under the node object executing the method.

A leaf object associated with the new leaf is returned.

The new leaf is inserted as the child of the current node, although there is no explicit ordering of leaves.

Note. To insert a leaf with a specific range, use the InsertChildLeaf method.

Parameters

None.

Returns

A leaf object associated with the new leaf. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWLEAF = &MYLNODE.InsertDynChildLeaf();

If None(&NEWLEAF) Then
   /* Do error processing */
End-If;
```

See Also

Chapter 43, "Tree Classes," InsertChildLeaf, page 2331

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None

InsertSib

Syntax

InsertSib*(NodeName)*

Description

The InsertSib method inserts a new node (as specified by *NodeName)* as a sibling node to the node object executing the method. The node specified by *NodeName* must be a *new* node. You receive an error if the node already exists.

A node object associated with the new node is returned. If the new node isn't inserted successfully, the method returns Null.

Parameters

Parameter	*Description*
NodeName	Specify a name for the new node. This parameter takes a string value. *NodeName* must not already exist.

Returns

A reference to the new node. If this method fails, it returns False.

Example

```
&NEWNODE = &MYNODE.InsertSib("100500");
```

InsertSibRecord

Syntax

InsertSibRecord*(NodeName)*

Description

The InsertSibRecord method inserts a new record (as specified by *NodeName)* as a sibling node of the parent node executing this method. This method works only on trees that are Query Access Trees.

A node object associated with the new node is returned, otherwise this method returns Null.

Parameters

Parameter	*Description*
NodeName	Specify an existing record name. This parameter takes a string value. This record must not already be a sibling node for the node executing the object (that is, a node can't have the same record as a node more than once.)

Returns

A reference to the new node. If this method fails, it returns False.

Example

```
&NEWNODE = &MYNODE.InsertSibRecord("PERSONAL_DATA");
```


LoadABNChart

Syntax

LoadABNChart*(&chart_rowset,&relactions_rowset, requested_node,initial_node)*

Description

Use this method to load the requested tree node into the SmartNavigation chart rowset and the component buffer.

Parameters

Parameter	***Description***
&chart_rowset	Specifies the SmartNavigation chart rowset. Typically, this is the rowset returned by the GetABNChartRowSet function.
&relactions_rowset	Specifies the related actions rowset. Typically, this is the rowset returned by the GetABNRelActnRowSet function.
requested_node	Specifies as a Boolean value whether the calling node is the requested node.
initial_node	Specifies the initial chart node. Typically, this is returned directly by calling the GetABNInitialNode function.

Returns

None.

Example

```
&MyNode.LoadABNChart(&chart_rs, &ra_rs, True, GetABNInitialNode(&reqParams));
&ChildNode.LoadABNChart(&chart_rs, &ra_rs, False, GetABNInitialNode(&reqParams));
```

See Also

Chapter 43, "Tree Classes," LoadABNChartOrdered, page 2338

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetABNChartRowSet; *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetABNInitialNode and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetABNRelActnRowSet

LoadABNChartOrdered

Syntax

LoadABNChartOrdered(*&chart_rowset,&relactions_rowset, requested_node, initial_node,order*)

Description

Use this method to load the requested tree node into the SmartNavigation chart rowset and the component buffer. The order for this node within its chart level is specified by this method.

Parameters

Parameter	***Description***
&chart_rowset	Specifies the SmartNavigation chart rowset. Typically, this is the rowset returned by the GetABNChartRowSet function.
&relactions_rowset	Specifies the related actions rowset. Typically, this is the rowset returned by the GetABNRelActnRowSet function.
requested_node	Specifies as a Boolean value whether the calling node is the requested node.
initial_node	Specifies the initial chart node. Typically, this is returned directly by calling the GetABNInitialNode function.
order	Specify the order for the node as a number. **Note.** For a given chart level, all chart leaves must have a display order greater than zero.

Returns

None.

Example

```
&MyNode.LoadABNChartOrdered(&chart_rs, &ra_rs, True, ⇒
GetABNInitialNode(&reqParams), 5);
&ChildNode.LoadABNChartOrdered(&chart_rs, &ra_rs, False, ⇒
GetABNInitialNode(&reqParams), 5);
```

See Also

Chapter 43, "Tree Classes," LoadABNChart, page 2337

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetABNChartRowSet; *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetABNInitialNode and *PeopleTools 8.51 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetABNRelActnRowSet

MoveAsChild

Syntax

MoveAsChild*(Node)*

Description

The MoveAsChild method moves the node executing the method to a different node in the tree. The node becomes the first child node under the named node. The specified node becomes the new parent to the child node. The node specified by *Node* must be an existing node in the current tree. All child nodes and details are also moved with the node.

Parameters

Parameter	***Description***
Node	Specify a node object. This parameter value must be an object, not a string or a name.

Note. To specify a node name, not a node object, use the MoveAsChildByName method.

Returns

A number; 0 if method is successful.

Example

The following example moves node 10200 from node 10100 (old parent) to node 00001 (new parent).

```
&MY_NODE = &MY_TREE.FindNode("10200", "");
&NEW_PARENT = &MY_TREE.FindNode("00001", "");
If &NEW_PARENT <> Null Then
   &MY_NODE.MoveAsChild(&NEW_PARENT);
End-If;
```


See Also

Chapter 43, "Tree Classes," MoveAsChildByName, page 2340

MoveAsChildByName

Syntax

MoveAsChildByName*(NodeName)*

Description

The MoveAsChildByName method moves the node executing the method to a different node in the tree. The node becomes the first child node under the parent node. The specified node becomes the new parent to the node. The node specified by *NodeName* must be a valid node name. All child nodes and details are also moved with the node.

Parameters

Parameter	***Description***
NodeName	Specify the name of a node. This parameter takes a string value. *NodeName* must be a valid node for the existing tree.

Note. To specify a node object, not a node name, use the MoveAsChild method.

Returns

A number; 0 if method is successful.

Example

The following moves node 10200 from node 10100 (old parent) to node 00001 (new parent)

```
&MY_NODE = &MY_TREE.FindNode("10200", "");
If &MY_NODE <> Null Then
   &MY_NODE.MoveAsChildByName("00001");
End-If;
```

See Also

Chapter 43, "Tree Classes," MoveAsChild, page 2339

MoveAsSib

Syntax

MoveAsSib*(Node)*

Description

The MoveAsSib method moves the node executing the method to a new place in the tree. The current node becomes the next sibling node under the specified node. All child nodes and details are also moved with the node.

Parameters

Parameter	*Description*
Node	Specify a node object. This parameter value must be an object, not a string or a name.

Note. To specify a node name, not a node object, use the MoveAsSibByName method.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("20000", "");
&MYNODE2 = &MYTREE.FindNode("20100", "");
&MYNODE.MoveAsSib(&MYNODE2);
```

See Also

Chapter 43, "Tree Classes," MoveAsSibByName, page 2341

MoveAsSibByName

Syntax

MoveAsSibByName*(NodeName)*

Description

The MoveAsSibByName method moves the node executing the method to a new place in the tree. The current node becomes the next sibling node under the specified node. All child nodes and details are also moved with the node.

Parameters

Parameter	***Description***
NodeName	Specify the name of a node. This parameter takes a string value. *NodeName* must be a valid node for the existing tree.

Note. To specify a node object, not a node name, use the MoveAsSib method.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("20000", "");
&MYNODE.MoveAsSibByName("10100");
```

See Also

Chapter 43, "Tree Classes," MoveAsSib, page 2341

PasteChild

Syntax

`PasteChild``()`

Description

The PasteChild method makes the node or leaf from the buffer (clipboard) a child to the node executing the method. You must use the Cut node or leaf method before you can paste a node or leaf.

You can have only one object at a time on the clipboard. If you cut another node or leaf, the node or leaf on the clipboard is overwritten.

Parameters

None.

Returns

A number; 0 if method is successful.

See Also

Chapter 43, "Tree Classes," Cut, page 2303; Chapter 43, "Tree Classes," PasteSib, page 2343 and Chapter 43, "Tree Classes," Cut, page 2327 node method

PasteSib

Syntax

```
PasteSib()
```

Description

The PasteSib method makes the node from the buffer (clipboard) a sibling to the node executing the method. You must use the Cut node method before you can paste a node.

You can have only one object at a time on the clipboard. If you cut another leaf or node, the node on the clipboard is overwritten.

Parameters

None.

Returns

A number; 0 if method is successful.

See Also

Chapter 43, "Tree Classes," Cut, page 2303; Chapter 43, "Tree Classes," PasteChild, page 2342 and Chapter 43, "Tree Classes," Cut, page 2327 node method

RefreshDescription

Syntax

`RefreshDescription``()`

Description

The RefreshDescription method enables you to change the description of a node on a page and have the update be displayed immediately. If you don't use RefreshDescription, you must save the tree and reopen it for the change to be displayed.

Parameters

None.

Returns

A zero (0) if description is refreshed successfully, a different error number otherwise.

Example

```
&result= &NodeObj.refreshdescription();
```

Rename

Syntax

`Rename``(`*`NodeName`*`)`

Description

The Rename method renames the node object executing the method without changing any other properties. The parameter *NodeName* takes a string value. The node specified by *NodeName* must be a new node. You receive an error if the node already exists.

Note. The Rename method does not rename the node in a user node component. Your application needs to do that.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("10900", "");
&MYNODE.Rename("10200");
```

SwitchLevel

Syntax

SwitchLevel(*NewLevelNumber*)

Description

The SwitchLevel methods enables you to move a node down from one level to another.

NewLevelNumber must specify a valid level number. For trees in which the Level Use parameter is set to Strictly Enforce Levels, the new level must be at least one level lower than the node's parent level.

If the level specified by *NewLevelNumber* doesn't exist, it's automatically generated and added to the tree.

Parameters

Parameter	***Description***
NewLevelNumber	Specify the level number to where you want the node moved.

Returns

A zero (0) if node is moved successfully, a different error number otherwise.

Unbranch

Syntax

Unbranch()

Description

The Unbranch method is the opposite of the Branch method: that is, it unbranches the node executing the method if it is branched. All of the child node and leaves of the node become part of the current open tree and accessible in that tree. You can't unbranch the Root Node in a branched tree.

If the node executing the method isn't branched, you receive a runtime error. Use the IsBranched property to make sure a node is branched.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("10900", "");
&MYNODE.Unbranch();
```

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Creating Trees," Working with Tree Branches

Node Class Properties

In this section, we discuss the Node class properties. The properties are discussed in alphabetical order.

AllChildCount

Description

This property returns the number of all the child nodes and leaves of the node, that is, all the nodes and leaves below this node.

This property is read-only.

AllChildNodeCount

Description

This property returns the number of child nodes of the node, that is, all the nodes below this node.

To determine all the leaves below a node, subtract the AllChildNodeCount from the AllChildCount

This property is read-only.

Example

To determine all the leaves below a node, use the AllChildNodeCount with the AllChildCount property.

```
&AllCount = (&MyNode.AllChildCount - &MyNode.AllChildNodeCount);
```


ChildLeafCount

Description

This property returns the number of immediate child leaves below this node.

This property is read-only.

ChildNodeCount

Description

This property returns the number of immediate child nodes below this node.

This property is read-only.

CollmageName

Description

This property enables you to specify the image name for a collapsed node.

This property is read-write.

Description

Description

This property returns the description of the node.

This property is read-only.

DisplayLevelNumber

Description

When using the Tree API to create a graphic representation of the tree (such as for HTML Tree Manager or other application pages) use this property to indicate the display level, that is, tell the user how many levels deep the node is. This is generally used for trees where levels aren't used and the LevelNumber property always returns 0.

This property is read-only.

See Also

Chapter 43, "Tree Classes," LevelNumber, page 2352

ExpImageName

Description

This property enables you to specify the expanded image name for a node.

This property is read-write.

FirstChildLeaf

Description

This property returns a reference to the first child leaf of the node. This is the leaf that appears highest in the list of children of the node in PeopleSoft Tree Manager.

This property is read-only.

Example

```
&NEWLEAF = &MYNODE.FirstChildLeaf;
```

FirstChildNode

Description

This property returns a reference to the first child node of the node. This is the node that appears highest in the list of children of the node in PeopleSoft Tree Manager.

This property is read-only.

Example

```
&CHILDNODE = &MYNODE.FirstChildNode;
```


HasChildLeaves

Description

This property returns True if the node has immediate child leaves, False otherwise.

This property is read-only.

HasChildNodes

Description

This property returns True if the node has immediate child nodes, False otherwise.

This property is read-only.

HasChildren

Description

This property returns True if the node has either child leaves or child nodes anywhere in the subtree, not just immediately under the node.

This property is read-only.

HasNextSib

Description

This property returns True if the node has a next sibling, that is, if it isn't the last node.

This property is read-only.

Example

```
While &MYNODE.HasNextSib
   &MYNODE = &MYNODE.NextSib;
   /* do some processing */
End-While;
```

HasPrevSib

Description

This property returns True if the node has a previous sibling, that is, if it isn't the first node.

This property is read-only.

IsBranched

Description

This property returns True if the node is branched, False otherwise.

This property is read-only.

IsChanged

Description

This property returns True if the node has been edited or changed but the current tree hasn't been saved.

This property is read-only.

IsCut

Description

This property returns True if the node has been cut from the displayed tree. This property is generally used with the HTML Tree Manager.

This property is read-only.

IsDeleted

Description

This property returns True if the node has been deleted but the current tree hasn't been saved.

This property is read-only.

IsInserted

Description

This property returns True if the node was inserted as a new node into the tree but the tree hasn't been saved.

This property is read-only.

Example

```
If &MYNODE.IsInserted Then
   &MYTREE.Save();
End-if;
```

IsRoot

Description

This property returns True for both of the following:

- the node is the root node of an unbranched tree.
- the node is the top node of an opened tree branch.

Otherwise, the property returns False.

This property is read-only.

LastChildLeaf

Description

This property returns a reference to the last child leaf of the node. This is the leaf that appears lowest in the list of children of the node in PeopleSoft Tree Manager.

This property is read-only.

Example

```
&NEWLEAF = &MYNODE.LastChildLeaf;
```


LastChildNode

Description

This property returns a reference to the last child node of the node. This is the node that appears lowest in the list of children of the node in PeopleSoft Tree Manager.

This property is read-only.

Example

```
&CHILDNODE = &MYNODE.LastChildNode;
```

LevelNumber

Description

This property returns the level number of the node. Values are 1-99 for Strict or Loose level trees. This property returns 0 for trees that don't have levels.

Note. PeopleSoft does not recommend using this property to set the level of the node. Instead, use the SwitchLevel method.

This property is read-write.

See Also

Chapter 43, "Tree Classes," SwitchLevel, page 2345

Name

Description

This property returns the name of the node (as a string.) You must set this property to a valid value if you are creating a new node.

Note. Do not use this property to change the name of an existing node. The change will not be reflected in the database. You must use the Rename method to change the name of an existing node.

This property is read-write.

See Also

Chapter 43, "Tree Classes," Rename, page 2344

NextSib

Description

This property returns a reference to the next sibling node. If there isn't a next sibling node, Null is returned.

This property is read-only.

Example

```
While &MYNODE.HasNextSib
   &MYNODE = &MYNODE.NextSib;
   /* do some processing */
End-While;
```

Parent

Description

This property returns a reference to the node that is the parent of the node executing the property. If the current node has no parent (is a root node) False is returned.

This property is read-only.

Example

```
&PARENT = &MYNODE.Parent;
```

PrevSib

Description

This property returns a reference to the node that is the previous sibling node to the node executing the property. If there isn't a previous sibling, Null is returned.

This property is read-only.

Example

```
If &MYNODE.PrevSib Then
   /* do processing */
End-if;
```


State

Description

This property returns the *state* of the node, that is, does it have children, and are they expanded or collapsed.

See Chapter 43, "Tree Classes," Expand, page 2328.

The values for this property are:

Value	*Description*
0	Node has no children
1	Children are expanded
2	Children are collapsed
3	Leaves are collapsed

This property is read-only.

Example

```
&VALUE = &MYNODE.State;
If &VALUE = 2 Then
   &MYNODE.Expand(0);
End-if;
```

TreeBranchName

Description

This property returns the branch name of the tree as a string if the tree is branched. If not branched, this property returns a blank string.

This property is read-only.

TreeEffDt

Description

This property returns the effective date of the tree as a string if the tree is effective-dated. If not effective-dated, this property returns a blank string.

This property is read-only.

TreeName

Description

This property returns the name of the tree as a string.

This property is read-only.

TreeSetId

Description

This property returns the SetID of the tree as a string if the tree has a SetID. If the tree doesn't have a SetID, this property returns a blank string.

This property is read-only.

TreeUserKeyValue

Description

This property returns the UserKeyValue of the tree as a string if the tree has a UserKeyValue. If the tree doesn't have a UserKeyValue, this property returns a blank string.

Type

Description

This property returns the type of the node. This property takes a string value. Values are:

- "G" (Group): normal unbranched node or record group
- "B" (Branched nodes)
- "R" (Query Record): used for Query Access Trees only

This property is read-only.

Tree Class

Tree objects are instantiated from a session object with the GetTree method.

The following code sample gets a tree, then opens the tree structure associated with that tree:

```
Local string &TREE_NAME;
Local string &TREE_DT;
Local ApiObject &MYSESSION, &VC_TREE, &STRUCT;


/* Get and Open the Tree using the Tree API */
&MYSESSION = %Session;

/* Get the Tree Name and Effective Date from the level 0 record */
&TREE_NAME = "CUSTOMER";
&TREE_DT = "1900-01-01";

/* Get and Open the Tree  */

&VC_TREE = &MYSESSION.GetTree();

/* open the tree with read access only */

&VC_TREE.OPEN("BKINV", "", &TREE_NAME, &TREE_DT, "", False);

/* Get and Open Tree Structure */

&STRUCT = VC_TREE.Structure;
```

See Chapter 43, "Tree Classes," GetTree, page 2301.

Tree Class Methods

In this section, we discuss the Tree class methods. The methods are discussed in alphabetical order.

Audit

Syntax

Audit()

Description

The Audit method audits the tree object executing the method to determine its validity.

The Audit method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can audit it.

Returns

A number: 0 if the tree passes all audits. If Audit returns a value not equal to 0, an error is logged.

See Also

Chapter 43, "Tree Classes," Open, page 2369

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

AuditByName

Syntax

```
AuditByName(SetID,UserKeyValue,TreeName, EffDt,BranchName)
```

Description

The AuditByName method audits the tree specified by the parameters passed to it. The AuditByName method can be used only with a tree identifier, it cannot be used on a fully instantiated, open tree. Before you use the AuditByName method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

Parameters

Parameter	*Description*
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.

Parameter	Description
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

A number: 0 if the tree passes all audits. If this method returns a value not equal to 0, an error is logged.

Example

```
&ISVALID = &MYTREE.AuditByName("", "", "PERSONAL_DATA", "05-05-1997", "");
```

See Also

Chapter 43, "Tree Classes," Close, page 2358 and Chapter 43, "Tree Classes," IndirectionMethod, page 2411

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

Close

Syntax

```
Close()
```

Description

The Close method closes the tree, freeing the memory associated with that object, and discarding any changes made to the tree since the last save. The Close method can be used only on an open tree, not a closed tree. This means you must have opened the tree with the Open method before you can close it. To save any changes, you must use the Save method before using Close.

Returns

A number: 0 if the method completed successfully.

See Also

Chapter 43, "Tree Classes," Open, page 2369

Copy

Syntax

```
Copy(FromSetId,FromUserKeyValue,FromTreeName, FromEffDt,FromBranchName,ToSetId,
ToUserKeyValue,ToTreeName,ToEffDt,ToBranchName)
```

Description

The Copy method copies from the tree identified by the *From* parameters and copies it to the tree identified with the *To* parameters. The tree specified by the *To* parameters must be a *new* tree. You receive an error if the tree already exists. The tree specified by the *From* parameters does *not* have to match the tree identifier executing the method.

The Copy method can be used only with a *closed* tree, it cannot be used on an open tree. Before you use the Copy method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

To access the new tree, you must use the Open method.

Parameters

Parameter	***Description***
FromSetId	Specify the table indirection key for the tree to be copied from. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
FromUserKeyValue	Specify the User Key Value for the tree to be copied from. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
FromTreeName	Specify the name for the tree to be copied from. This parameter takes a string value.
FromEffDt	Specify the effective date for the tree to be copied from. This parameter takes a string value.
FromBranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Parameter	*Description*
ToSetId	Specify the table indirection key for the tree to be copied to. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
ToUserKeyValue	Specify the User Key Value for the tree to be copied to. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B" you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
ToTreeName	Specify the name for the tree to be copied to. This parameter takes a string value.
ToEffDt	Specify the effective date for the tree to be copied to. This parameter takes a string value.
ToBranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

An integer: 0 if copied successfully.

Example

```
&MYTREE.Copy("","","PERSONAL_DATA", "05-05-1997", "", "","","PERSONAL_EDIT1", "05-⇒
05-1997", "");
```

See Also

Chapter 43, "Tree Classes," Close, page 2358; Chapter 43, "Tree Classes," Open, page 2369 and Chapter 43, "Tree Classes," IndirectionMethod, page 2411

Create

Syntax

Create*(SetID,UserKeyValue,TreeName, EffDt,StructureName)*

Description

The Create method creates a new tree, based on the parameters passed with the method. The specified tree must be a new tree. You receive an error if the tree already exists.

The Create method can be used only with a closed tree, it cannot be used on an open tree. Before you use the Create method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

After you create a new tree, you don't have to open it with the Open method. The existing tree object points to the new tree.

The tree structure specified with *StructureName* determines what database fields the new tree is based on, what pages you can use to create tree nodes and detail values, and what record definitions PeopleSoft Tree Manager saves tree-related data in. The tree structure must already exist.

The new tree must be saved (Save, SaveAs, and so on) for this tree to be saved to the database.

Parameters

Parameter	***Description***
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
StructureID	Specify the name of the tree structure to use for this tree. This parameter takes a string value.

Returns

An integer: 0 if created successfully.

Example

```
&MYTREE.Create("","","PERSONAL_NEW", "05-05-1997", "PERSONAL_DATA");
```

See Also

Chapter 43, "Tree Classes," Close, page 2358; Chapter 43, "Tree Classes," IndirectionMethod, page 2411 and *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager*, "Creating Trees"

Delete

Syntax

Delete(*SetID,UserKeyValue,TreeName, EffDt,BranchName*)

Description

The Delete method deletes the specified tree from the database. The Delete method can be used only with a closed tree, it cannot be used on an open tree. Before you use the Delete method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

The specified tree does not have to match the tree identifier executing the method.

Parameters

Parameter	***Description***
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.

Parameter	Description
BranchName	Specify the name of the branch for the tree. This parameter takes a string value. If the tree is unbranched, you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

An integer: 0 if deleted successfully.

Example

```
&MYTREE=&Session.GetTree();

If ALL(&MYTREE) Then
   &RETVALUE = &MYTREE.Delete("","","PERSONAL_REN2", "05-05-1997", "");
End-If;
```

See Also

Chapter 43, "Tree Classes," Close, page 2358 and Chapter 43, "Tree Classes," IndirectionMethod, page 2411

Exists

Syntax

Exists*(SetID,UserKeyValue,TreeName, EffDt,BranchName)*

Description

The Exists method finds an existing tree, as specified by the parameters. The parameters must specify a unique tree. If a tree matching the parameters is found, the method returns 0. This method works with either open or closed trees.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Parameter	***Description***
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

An integer: 0 if tree exists.

Example

```
&RSLT = &MYTREE.Exists("", "", "VENDOR_PER_DATA", "05-03-1998", "");

If &RSLT = 0 Then
   /* tree exists, do processing */
Else
   /* tree doesn't exist, do error processing */
End-if;
```

See Also

Chapter 43, "Tree Classes," IndirectionMethod, page 2411

FindLeaf

Syntax

`FindLeaf`*`(RangeFrom,RangeTo)`*

Description

The FindLeaf method finds an existing leaf (specified by the *range* parameters) in the tree executing this method. If the leaf is found, a leaf object is returned. If the leaf isn't found, a Null is returned.

You can use wildcards with the *range* parameters. Use an asterisk (*) in place of a number of characters. Use a question mark (?) in place of one character. You can also provide one of the ranges, and specify a Null string for the other.

Suppose you wanted to find the leaf with the range 81005 to 82000. The following would be valid:

```
&MyLeaf = &MyTree.FindLeaf("81*", "82000");
```

This would also be valid:

```
&MyLeaf = &MyTree.FindLeaf("8100?", "82000");
```

You could also specify this:

```
&MYLEAF = &MYTREE.FindLeaf("81000", "");
```

If you specify the exact same value for both *RangeFrom* and *RangeTo,* the *RangeTo* parameter is ignored.

Parameters

Parameter	***Description***
RangeFrom	Specify the starting range of the leaf to be found. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf to be found. This parameter takes a string value.

Returns

A reference to a leaf object.

Example

```
&MYLEAF = &MYTREE.FindLeaf("81000", "");
```

FindNode

Syntax

FindNode*(NodeName,Description)*

Description

The FindNode method returns a reference to the node object specified by the parameters passed to the method. If the node is found, a node object is returned. If the node isn't found, a Null is returned.

Note. If you've only specified a node name, and the node you're searching for is in an unexpanded portion of the tree, this method expands the tree down to that node. If you've only specified a node description the search, occurs only in the memory. If the node isn't found, the tree is not expanded.

You should specify either the node name other description, not both. You should specify a Null string for the other. If both are provided, only the node name is used and the description is ignored.

You can use wildcards with the parameters. Use an asterisk (*) in place of a number of characters. Use a question mark (?) in place of one character.

Suppose you wanted to find the node called 10200, with a description of "Human Resources". The following would be valid:

```
&MyNode = &MyTree.FindNode("", "Human*");
```

This would also be valid:

```
&MyNode = &MyTree.FindNode("1020?", "");
```

You could also use the following:

```
&MYNODE = &MYTREE.FindNode("10200", "");
```

Note. If you're searching using wildcared characters and more than one node matches the search conditions, only the first occurence of the node is returned

Parameters

Parameter	***Description***
NodeName	Specify the name of the node that you want to find. This parameter takes a string value. Specify either the node name or the description.
Description	Specify the description of the node that you want to find. This parameter takes a string value. Specify either the node name or the description.

Returns

A reference to a node object.

Example

```
&MYNODE = &MYTREE.FindNode("10200", "");
```

FindRoot

Syntax

FindRoot()

Description

The FindRoot method returns a reference to the root node object of the tree object executing the method. This method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can find the root node.

This method returns an error if the tree has no nodes associated with it.

Returns

A reference to a node object.

Example

```
&MYNODE = &MYTREE.FindRoot();
```

See Also

Chapter 43, "Tree Classes," Open, page 2369

InsertRoot

Syntax

`InsertRoot`*`(NodeName)`*

Description

The InsertRoot method inserts a node at the top of the tree object executing the method. *NodeName* is the name of the node you are creating. This parameter takes a string value. The InsertRoot method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can insert any nodes.

You can insert the root node only in a new tree, that is, a tree that was just created with the Create method. Also, after you create a new tree, you must insert the root node before you can insert any other nodes.

In a strict level tree, the first level must be defined before you can insert the root node.

This method returns an error if the tree already has nodes in it.

Returns

A reference to a node object.

Example

```
&MyRootNode = &MYTREE.InsertRoot("000001");
```

See Also

Chapter 43, "Tree Classes," Open, page 2369

LeafExists

Syntax

LeafExists*(RangeFrom,RangeTo)*

Description

The LeafExists method enables you to verify if the leaf specified by the parameters exists in the current tree.

Suppose you wanted to check the leaf with the range 81005 to 82000 exists. The following would be valid:

```
&MyResult = &MyTree.LeafExists("81005", "82000");
```

This would also be valid, and would check on leaves up to 82000:

```
&MyResult = &MyTree.LeafExists("", "82000");
```

If you specify the exact same value for both *RangeFrom* and *RangeTo,* the *RangeTo* parameter is ignored.

Parameters

Parameter	***Description***
RangeFrom	Specify the starting range of the leaf you're verifying. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf you're verifying. This parameter takes a string value.

Returns

An integer: 0 if the leaf exists.

See Also

Chapter 43, "Tree Classes," NodeExists, page 2375 and Chapter 43, "Tree Classes," FindLeaf, page 2364

LockTree

Syntax

LockTree()

Description

Use the LockTree method to check out a tree for editing. Use this method to prevent saving conflicts when you have multiple users accessing the same tree, using both the Tree API and PeopleSoft Tree Manager. To release a tree, use the UnlockTree method.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

To verify if a tree is already locked, use the LockStatus property.

This method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method.

Parameters

None.

Returns

An integer: 0 if locked successfully.

See Also

Chapter 43, "Tree Classes," TreeLocksNumber, page 2382; Chapter 43, "Tree Classes," UnlockTree, page 2383; Chapter 43, "Tree Classes," HasLockedBranches, page 2389; Chapter 43, "Tree Classes," LockStatus, page 2396 and Chapter 43, "Tree Classes," UseUpdateReservation, page 2402

Open

Syntax

Open(*SetID, UserKeyValue, TreeName, EffDt, BranchName, Update*)

Description

The Open method opens the tree object specified by the parameters. The Open method can be used only with a closed tree, it cannot be used on an open tree. You cannot read or set any properties of a tree until after you open it. When you open a tree, you can specify whether you want to open it for update, or in read-only mode, with the *Update* parameter. If you try updating or writing to a tree opened in read-only mode, you receive an error.

Note. Because EFFDT is a key field for a tree definition, an exact match is required to open a tree with the Open method. A less restrictive method, OpenAsOfDate, provides an alternative to Open by specifying an *AsOfDate* parameter instead of the *EffDt* parameter.

Parameters

Parameter	***Description***
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for the tree. This parameter takes a string value.
EffDt	Specify the effective date for the tree. This parameter takes a string value. The format must be in the correct format for the database platform the code is executing on.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
Update	Specify if you want to open the tree for update or in read-only mode. This parameter takes a Boolean value. Values for *Update* are: • True: open tree structure in update mode (read-write). • False: open tree structure in read-only mode. If you try updating or writing to a tree structure opened in read-only mode, you receive an error.

Returns

An integer: 0 if opened successfully.

Example

The following example opens a tree object, then tests to see if the tree was actually opened:

```
&RSLT = &MYTREE.Open("","","PERSONAL_DATA", "05-05-1997", "", True);

If &RSLT = 0 Then
   /* normal processing */
Else
   /* error processing tree isn't open */
End-if;
```

See Also

Chapter 43, "Tree Classes," Close, page 2358 and Chapter 43, "Tree Classes," OpenAsOfDate, page 2371

Chapter 43, "Tree Classes," IndirectionMethod, page 2411

OpenAsOfDate

Syntax

OpenAsOfDate*(SetID,UserKeyValue,TreeName*, *AsOfDate,BranchName,Update)*

Description

The OpenAsOfDate method operates similarly to the Open method and uses the same parameters except that instead of requiring the *EffDt* parameter, OpenAsOfDate uses a less restrictive *AsOfDate* parameter. The OpenAsOfDate method opens the most recent tree based on *AsOfDate.* If *AsOfDate* is not supplied, the current date is used.

Note. Because EFFDT is a key field for a tree definition, an exact match is required to open a tree with the Open method.

The OpenAsOfDate method can be used only with a closed tree; it cannot be used on an open tree. You cannot read or set any properties of a tree until after you open it. When you open a tree, you can specify whether you want to open it for update, or in read-only mode, with the Update parameter. If you try updating or writing to a tree opened in read-only mode, you receive an error.

Parameters

Parameter	***Description***
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for the tree. This parameter takes a string value.
AsOfDate	Specify the "as-of-date" for the tree as a string. The format must be in the correct format for the database platform the code is executing on. If this parameter is not supplied, then the current date is used. If more than one tree matches the *SetID*,*UserKeyValue*,*TreeName*, and *BranchName* parameters, the tree with most recent effective date less than or equal to the as-of-date is opened.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
Update	Specify if you want to open the tree for update or in read-only mode. This parameter takes a Boolean value. Values for *Update* are: • True: open tree structure in update mode (read-write). • False: open tree structure in read-only mode. If you try updating or writing to a tree structure opened in read-only mode, you receive an error.

Returns

An integer: 0 if opened successfully.

Example

The following example opens a tree object with most recent effective date matching the other tree criteria. Then, the program tests to see if the tree was actually opened:


```
&ret = &MyTree.OpenAsOfDate("QEDM1","","PERSONAL_DATA", "", "", True);

If &ret = 0 Then
   /* normal processing */
Else
   /* error processing tree isn't open */
End-if;
```

See Also

Chapter 43, "Tree Classes," Open, page 2369

OpenForExport

Syntax

OpenForExport(*SetID*, *UserKeyValue*, *TreeNameEffDt*)

Description

Use the OpenForExport method to open a tree for export processes (using TreeMover.)

Parameters

Parameter	***Description***
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for the tree. This parameter takes a string value.
EffDt	Specify the effective date for the tree. This parameter takes a string value. The format must be in the correct format for the database platform the code is executing on.

Returns

An integer: 0 if opened successfully.

Example

```
If &TREE.OpenForExport(&SETID, &USERKEYVALUE, &TREE_NAME, &TREE_EFFDT) <> 0 Then

/* processing */

End-if;
```

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Using TreeMover," File Layout Details

OpenWholeTree

Syntax

OpenWholeTree(*SetID*,*UserKeyValue*,*TreeName*, *EffDt*)

Description

The OpenWholeTree method enables you to open a tree as if it were an unbranched tree. With this kind of tree, branched nodes cannot be distinguished from unbranched nodes. This means you can traverse the entire tree. It also means that branches have a node icon, not a branch icon. This method can be used only with a closed tree, it cannot be used on an open tree. You cannot read any properties of a tree until after you open it. This tree is always opened as read-only.

Note. You can't open a tree using this method and update it. The tree is opened in read-only mode.

Parameters

Parameter	***Description***
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Parameter	**Description**
UserKeyValue	Specify the User Key value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for the tree. This parameter takes a string value.
EffDt	Specify the effective date for the tree. This parameter takes a string value. The format must be in the correct format for the database platform the code is executing on.

Returns

An integer: 0 if opened successfully.

Example

```
&pSession = %Session;

&pTreeObj = &pSession.GetTree();
&status = &pTreeObj.OpenWholeTree(&Setid, &SetCntrlValue, &TreeName, String(&Eff⇒
Dt));
```

See Also

Chapter 43, "Tree Classes," Close, page 2358 and Chapter 43, "Tree Classes," IndirectionMethod, page 2411

NodeExists

Syntax

`NodeExists`*`(NodeName)`*

Description

The NodeExists method enables you to verify if the node specified by *NodeName* exists in the current tree.

Note. If the node you're searching for is in an unexpanded portion of the tree, this method expands the tree down to that node.

Parameters

Parameter	*Description*
NodeName	Specify the name of the node that you want to find. This parameter takes a string value.

Returns

An integer: 0 if the node exists.

See Also

Chapter 43, "Tree Classes," FindLeaf, page 2364; Chapter 43, "Tree Classes," FindNode, page 2365 and Chapter 43, "Tree Classes," LeafExists, page 2368

Rename

Syntax

```
Rename(FromSetId,FromUserKeyValue,FromTreeName, FromEffDt,FromBranchName,
ToTreeName)
```

Description

The Rename method renames a tree specified with the *From* parameters to the tree specified with *ToTreeName.* The tree specified by *ToTreeName* must be a new tree. You receive an error if the tree already exists. The tree specified by the *From* parameters does not have to match the tree executing the method.

The Rename method can be used only with a closed tree, it cannot be used on an open tree. Before you use the Rename method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

To access the new tree, you must use the Open method.

Parameters

Parameter	*Description*
FromSetId	Specify the table indirection key for the tree to be copied from. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Parameter	***Description***
FromUserKeyValue	Specify the User Key Value for the tree to be copied from. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B" you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
FromTreeName	Specify the name for the tree to be copied from. This parameter takes a string value.
FromEffDt	Specify the effective date for the tree to be copied from. This parameter takes a string value.
FromBranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
ToTreeName	Specify the new name of the tree. This parameter takes a string value.

Returns

An integer: 0 if renamed successfully.

Example

```
&MYTREE.Rename("","","PERSONAL_DATA3", "05-05-1997","", "PERSONAL_REN2");
```

See Also

Chapter 43, "Tree Classes," Close, page 2358; Chapter 43, "Tree Classes," Open, page 2369; Chapter 43, "Tree Classes," Close, page 2358 and Chapter 43, "Tree Classes," IndirectionMethod, page 2411

Save

Syntax

```
Save()
```

Description

The Save method writes any changes to the tree executing the method to the database. It also performs audits.

The Save method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can save it.

The tree object remains open after executing Save. You must execute the Close method on the object before it is closed and the memory freed.

This method returns an optional Boolean value: True if the tree was saved with no errors, False if there are one or more errors.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

Parameters

None.

Returns

An integer: 0 if save successfully.

SaveAs

Syntax

SaveAs*(SetID,UserKeyValue,TreeName, EffDt,BranchName)*

Description

The SaveAs method writes any changes to the tree executing the method to the new tree specified with the parameters. It also performs audits.

The tree specified with the parameters must be a new tree. You receive an error if the tree already exists. The new tree must have at least one different key field from the original tree.

The SaveAs method automatically closes the tree that is associated with the tree object executing the method, and associates that tree object with the new tree. Any changes made to the original tree since the last time the tree was saved are discarded. For example, suppose you opened a tree named DEPENDENT_BENEF and associated it with the variable &MYTREE. If you executed the SaveAs method with &MYTREE, changing the name to DEPENDENT_BENEF2, &MYTREE would now be associated with DEPENDENT_BENEF2, and the original tree, DEPENDENT_BENEF, would be closed.

```
&MYTREE.Open("", "", "DEPENDENT_BENEF", "02-02-1999", "", False);
/* do some processing */
&MYTREE.SaveAs("", "", "DEPENDENT_BENEF2", "02-05-1999", "");
```

&MYTREE is now associated with DEPENDENT_BENEF2, not with the original open tree.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

Parameters

Parameter	*Description*
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

This method returns an integer: 0 if saved successfully, a number greater than 0 otherwise.

Example

```
&MYTREE.SaveAs("", "", "DEPENDENT_BENEF2", "02-05-1999", "");
```

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

SaveAsDraft

Syntax

SaveAsDraft*(SetID,UserKeyValue,TreeName, EffDt,Branch`Name)*

Description

The SaveAsDraft method writes any changes to the tree executing the method to the new tree specified with the parameters. However, it does not perform audits. Trees that are saved in this way are marked as "draft" and will not be valid (that is, cannot be used with other PeopleTools) until the tree audit is passed during Save, SaveAs, or the Tree Audits Application Engine Tree Utility program..

The tree specified with the parameters must be a new tree. You receive an error if the tree already exists. The new tree must have at least one different key field from the original tree.

The SaveAsDraft method automatically closes the tree that is associated with the tree object executing the method, and associates that tree object with the new tree. Any changes made to the original tree since the last time the tree was saved are discarded.

See SaveAs for more details.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a Commit.

Parameters

Parameter	***Description***
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

This method returns an integer: 0 if saved successfully, a number greater than 0 otherwise.

Example

```
&MYTREE.  SaveAsDraft("","","PERSONAL_EDIT2", "05-05-1997", "");
```

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

SaveDraft

Syntax

SaveDraft()

Description

The SaveDraft method writes any changes to the tree executing the method to the database. However, it does not perform audits. Trees that are saved in this way are marked as "draft" and will not be valid (that is, cannot be used with other PeopleTools) until the tree audit is passed.

The SaveDraft method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can save it.

The tree object remains open after executing SaveDraft. You must execute the Close method on the object before it is closed and the memory freed.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a Commit.

Parameters

None.

Returns

This method returns an integer: 0 if saved successfully, a number greater than 0 otherwise.

See Also

Chapter 43, "Tree Classes," Close, page 2358 and Chapter 43, "Tree Classes," Open, page 2369

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

SetImportMode

Syntax

`SetImportMode(Nodes_NumberLeaves_Number)`

Description

Use the SetImportMode method to set up special memory allocation and simplified processing rules when running Tree Import.

This special mode also cuts back on the number of SQL calls by not getting node descriptions. This mode also blocks internal validation while inserting new node or leaf from an input file. All validation occurs during saving.

Parameters

Parameter	***Description***
Nodes_Number	Specify the number of nodes.
Leaves_Number	Specify the number of leaves.

Returns

An integer: 0 if set successfully.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Using TreeMover," File Layout Details

TreeLocksNumber

Syntax

`TreeLocksNumber(setID,UserKeyValue,TreeName,EffDt)`

Description

Use the TreeLocksNumber method to determine how many users have checked out any branches of a current tree. This method should be used to check if a tree definition could be changed without affecting other users working in different branches of a tree.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

You can use the TreeLocksNumber method with a both an open as well as a closed tree.

Parameters

Parameter	***Description***
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.

Returns

The number of locked branches. If the tree is not branched, but is checked out, the return value is 1.

See Also

Chapter 43, "Tree Classes," LockTree, page 2369; Chapter 43, "Tree Classes," UnlockTree, page 2383; Chapter 43, "Tree Classes," UnlockTree, page 2383 and Chapter 43, "Tree Classes," UseUpdateReservation, page 2402

UnlockTree

Syntax

```
UnlockTree()
```


Description

Use the UnlockTree method to release a tree after editing. Use this method to prevent saving conflicts when you have multiple users accessing the same tree, using both the Tree API and PeopleSoft Tree Manager. To checkout a tree, use the LockTree method.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

To verify if a tree is checked out, use the LockStatus property.

This method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method.

Parameters

None.

Returns

An integer: 0 if the tree is released successfully.

See Also

Chapter 43, "Tree Classes," LockTree, page 2369; Chapter 43, "Tree Classes," LockStatus, page 2396 and Chapter 43, "Tree Classes," UseUpdateReservation, page 2402

UpdateLock

Syntax

`UpdateLock``()`

Description

Use the UpdateLock method to update the timestamp representing time when the user who has the tree checked out performed the last action on a tree.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

Use this method to indicate that user who has the tree checked out is still working with it. The tree won't be released unless the Tree Inactivity Period specified on the People Tools Options page is expired.

This method should be called from PeopleCode in a response to any user action happened on a Tree Manager page.

This method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method.

Parameters

None.

Returns

An integer: 0 if activity time was updated successfully.

See Also

Chapter 43, "Tree Classes," LockTree, page 2369; Chapter 43, "Tree Classes," UnlockTree, page 2383 and Chapter 43, "Tree Classes," UseUpdateReservation, page 2402

Tree Class Properties

In this section, we discuss the Tree class properties. The properties are discussed in alphabetical order.

AllValues

Description

When the tree is audited, the AllValues property specifies whether the audit should verify that the tree includes all user data detail values. This property returns a Boolean value: True, check all values, False, do not check all values.

If this is a new tree, and you do not set this property, a default value of False is automatically set.

This property can be used only with an open tree, that is, you must use the Open method on the tree before you can use this property.

This property is read-write.

Example

```
&MyTree.AllValues = False;
```

AuditDetails

Description

Set this property to False in cases when you do not want to perform Detail Audit. It could happend if you are reusing existing structure that contains Detail information for tree that does not use details('Winter' tree).

The default value for this property is True.

This property is read-write.

Branches

Description

This property returns a reference to a branch collection object. This property is used only with branched trees: it returns Null for trees that aren't branched.

This property can be used only with an open tree, that is, you must use the open the tree using the Open method before you can use this property.

This property is read-only.

BranchImageName

Description

Use this property to specify the image used to display branches on a page. This property takes a string value.

This property is read-write.

BranchLevel

Description

This property returns the level number (as a number) where this branch occurs in the tree. This property is used only with trees with levels: it returns zero for trees that don't have levels.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
&LevelNumber = &MyTree.BranchLevel;
```

BranchName

Description

This property returns the name of the specific branch for the tree, as a string. This property is used only with branched trees: it returns an empty string for trees that aren't branched.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
&BranchName = &MyTree.BranchName;
```

Category

Description

This property returns the category name for the tree, as a string. The category is a high-level grouping under which you can organize your tree structures and tree definitions. If you are creating a new tree, this property is required.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Example

```
&MyTree.Category = "PurchaseOrders";
```

Description

Description

This property returns the description of the tree, as a string. If you are creating a new tree, this property is required.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Example

```
&MyTree.Description = "Departmental Security";
```

DuplicateLeaves

Description

In most trees, you want each detail value to appear only once in the tree. To permit duplicate values, set this property to True.

If this is a new tree, and you do not set this property, a default value of False is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

EffDt

Description

This property returns the effective date of the tree, as a date.

Note. PeopleSoft does not recommnd using this property to change data, as changes are not stored in the database.

If this is a new tree, and you do not set this property, a default value of the current date is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Example

```
&TreeEffDt = &MYTREE.EffDt;
```


HasDetailRanges

Description

This property returns True if the tree has any detail ranges.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

HasLockedBranches

Description

This property indicates whether any user other then current one has checked out any branch in a tree. This property returns a Boolean value: true if the tree has checked out branches, false otherwise.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

IsBranched

Description

This property indicates whether the tree is branched. Possible values for this property are True and False: True if the tree is branched, False if it isn't.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
If &MyTree.IsBranched Then
   /* Do branched processing */
Else
   /*Do unbranched processing */
End-If;
```

IsChanged

Description

This property returns True if the tree has been changed but not saved, False if the tree is unchanged.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
If &MyTree.IsChanged Then
   &MyTree.Save();
End-if;
&MyTree.Close();
```

IsOpen

Description

This property returns True if the tree is open, False if the tree is closed.

This property is read-only.

Example

```
If Not &MyTree.IsOpen Then
   &MyTree.Open("","","PERSONAL_DATA", &Date, "", True);
End-If;
```

IsQueryTree

Description

This property returns True if the tree is a Query access tree. It returns False if the tree is not a Query Access Tree.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

IsValid

Description

This property returns True if the tree is a valid tree that has passed all audits. It returns False otherwise.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

IsVersionChanged

Description

This property indicates if a tree version has been changed in the database. This property returns a Boolean value, true if the version was changed, false otherwise.

Use this function in a multi-user environment to check if a tree loaded in a read-only mode by User1 should be reloaded. A possible reason why it needs to be reloaded could be triggered by tree version change after User2, who had this tree checked out, released it.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

IsWholeTree

Description

This property indicates whether the entire tree is accessible. This property is set to True if a tree was opened using OpenWholeTree method. The OpenWholeTree method opens a tree in read-only mode without paying attention to the existing tree branches.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
If &MyTree.IsWholeTree Then
   /* Do processing */
Else
   /* close tree and reopen using OpenWholeTree method */
End-If;
```

KeyBranchName

Description

This property returns the branch name of the tree as a string if the tree is branched. If not branched, this property returns a blank string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the KeyBranchName property to get the tree branch name

This property is read-only.

KeyEffDt

Description

This property returns the effective date of the tree as a string if the tree is effective-dated. If not effective-dated, this property returns a blank string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the KeyEffDt property to get the tree effective date

This property is read-only.

KeyName

Description

This property returns the name of the tree as a string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the Name property to get the tree name.

This property is read-only.

KeySetId

Description

This property returns the SetID of the tree as a string if the tree has a SetID. If the tree doesn't have a SetID, this property returns a blank string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the KeySetId to get the tree Set Id.

This property is read-only.

KeyUserKeyValue

Description

This property returns the UserKeyValue of the tree as a string if the tree has a UserKeyValue. If the tree doesn't have a UserKeyValue, this property returns a blank string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the KeyUserKeyValue to get the tree UserKeyValue.

This property is read-only.

LeafCount

Description

This property returns the number of leaves in the tree.

This property can be used only with an open tree, that is, you must use the Open method on the tree before you can use this property.

This property is read-only.

LeafImageName

Description

Use this property to specify the image used to display leaves on a page. This property takes a string name.

This property is read-write.

LeafOnClipboard

Description

Use this property to determine if a leaf has been cut and is available to be 'pasted' into tree. This property returns a Boolean value: True if a leaf object exists on the clipboard, False otherwise.

This property is read-only.

LevelCount

Description

This property returns the number of levels defined for the tree. If the tree doesn't have any levels, this property returns zero (0).

See the LevelUse property.

Valid value range is a number between 0 and 99.

This property can only be used with an open tree, that is, you must use the Open method on the tree before you can use this property.

This property is read-only.

See Also

Chapter 43, "Tree Classes," LevelUse, page 2395

Levels

Description

This property returns a reference to a level collection object. This property is used only with trees that have levels: it returns Null for trees that don't have levels.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

See Also

Chapter 43, "Tree Classes," Level Collection Methods, page 2320

LevelUse

Description

This property returns the level use of the tree. The values are:

Value	*Description*
"S"	Strict levels
"L"	Loose levels
"N"	No levels

If this is a new tree, and you do not set this property, the default value is Strict Levels.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Example

```
If &MyTree.LevelUse = "S" Then
/* add levels */
End-If;
```

LockOwner

Description

This property returns the user ID of the user who has checked out a specific tree or a tree branch, as a string.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

LockStatus

Description

This property indicates if an opened tree is checked out.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

Possible values are:

Value	*Description*
0	Tree is not checked out by any user.
1	Tree is checked out by another user.
2	Tree is checked out by the current user and is editable.

This property is read-only.

See Also

Chapter 43, "Tree Classes," LockTree, page 2369; Chapter 43, "Tree Classes," TreeLocksNumber, page 2382 ; Chapter 43, "Tree Classes," UnlockTree, page 2383 and Chapter 43, "Tree Classes," LockOwner, page 2395

Name

Description

This property returns the name of the tree, as a string.

Note. Do not use this property to change the name of an existing tree. The change will not be reflected in the database. You must use the Rename method to change the name of an existing tree.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Example

```
&TreeName = &MyTree.Name;
```


See Also

Chapter 43, "Tree Classes," Rename, page 2376

NodeColImageName

Description

Use this property to specify the image displayed for a collapsed node. This property takes a string value.

This property is read-write.

NodeCount

Description

This property returns the number of nodes for the tree. A valid tree must have at least one node.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

NodeExpImageName

Description

Use this property to specify the displayed image of an expanded node. This property takes a string value.

This property is read-write.

NodeOnClipboard

Description

Use this property to determine if a node has been cut and is available to be 'pasted' into a tree. This property returns a Boolean value: True if a node object exists on the clipboard, False otherwise.

This property is read-only.

ParentLevel

Description

This property returns the number of the parent branch level in a branched tree. If the tree does not have levels or is an unbranched tree, zero (0) is returned. Valid value range is a number between 0 and 100.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

ParentName

Description

This property returns the name of the parent branch in a branched tree, as a string. If the tree does not have levels, is an unbranched tree, or if the tree is the root level, Null is returned.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

PerformanceMethod

Description

This property sets the method to use for select and join statements used to generate SQL with the tree. This property is used with nVision layouts and Query.

The values for this property are:

Value	***Description***
"L"	Suppress Join: Use Literal Values.
"S"	Sub-SELECT Tree Selector.
"J"	Join to Tree Selector.
"D"	User Application Defaults.

If this is a new tree, and you do not set this property, the default value "D" is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager, "Introduction to PeopleSoft Tree Manager" and *Enterprise PeopleTools 8.51 PeopleBook: PS/nVision*, "Understanding PS/nVision"

PerformanceSelector

Description

This property selects the tree selectors for nVision for the tree. This property is used with nVision layouts and Query.

The values for this property are:

Value	*Description*
"D"	Dynamic Selectors
"S"	Static Selectors

If this is a new tree, and you do not set this property, the default value "S" is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PS/nVision, "Understanding PS/nVision"

PerformanceSelectorOption

Description

This property selects the tree selector options for nVision for the tree. This property is used with nVision layouts and Query.

The values for this property are:

Value	Description
"S"	Single values
"R"	Ranges of values (>=…<=).
"B"	Range of values (BETWEEN).

If this is a new tree, and you do not set this property, the default value "R" is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PS/nVision, "Understanding PS/nVision"

SetID

Description

This property returns the SetID for the tree, as a string. This property is used only when the IndirectionMethod property for the tree structure the tree is based on has been set to "S". Otherwise, this property returns Null.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Note. PeopleSoft does not recommend setting the SetID for a tree, as there is no way to save this value to the database.

See Also

Chapter 43, "Tree Classes," IndirectionMethod, page 2411

Status

Description

This property returns the effective status of the tree. The values are:

Value	*Description*
"A"	Active
"I"	Inactive

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Structure

Description

This property returns a reference to the tree structure of the tree. Before you can use some of the methods or any of the properties of a tree structure, you must use the Open method to open the tree structure.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

StructureName

Description

This property returns the name *(structure ID)* of the tree structure the tree is based on, as a string. This property can be set only with a new tree.

Note. Even though this is a writable property, you will receive an error if you try to change the tree structure for an existing tree.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

See Also

Chapter 43, "Tree Classes," SetID, page 2400

TreeImageName

Description

Use this property to specify the displayed image for a tree. This property takes a string value.

This property is read-write.

UserKeyValue

Description

Use this property to read the key field value for the Node User Key Field specified from the NodeRecord on the associated tree structure.

This property is used only when the IndirectionMethod property for the tree structure has been set to "U" or "B". Otherwise, this property returns Null.

This property is read-write.

Note. PeopleSoft does not recommend setting the UserKeyValue for a tree, as there is no way to save this value to the database.

See Also

Chapter 43, "Tree Classes," NodeRecord, page 2417 and Chapter 43, "Tree Classes," IndirectionMethod, page 2411

UseUpdateReservation

Description

This property indicates if multi-user environment features are used. This property returns a Boolean value, true if multi-user environments features are used, false otherwise.

This property can be used with either an open or a closed tree.

This property is read-only.

Tree Structure Class

Tree structure objects are instantiated from the following:

- From a session object with the GetTreeStructure method.
- From a tree object with the Structure property.

See Chapter 43, "Tree Classes," GetTreeStructure, page 2301 and Chapter 43, "Tree Classes," Structure, page 2401.

Tree Structure Class Methods

In this section, we discuss the Tree Structure class methods. The methods are discussed in alphabetical order.

Close

Syntax

`Close``()`

Description

The Close method closes the tree structure object executing the method, freeing the memory associated with that object, and discarding any changes made to the tree structure. The Close method can be used only on an open tree structure, not on a closed tree structure. This means you must have opened the tree structure with the Open method before you can save it. To save any changes, you must use the Save method before using Close.

Returns

A number; 0 if the method completed successfully.

Copy

Syntax

`Copy``(FromStructId,ToStructId)`

Description

The Copy method copies the tree structure specified with *FromStructId* to the tree structure specified with *ToStructId.* Both parameters take string values. The tree structure specified by *ToStructId* must be a *new* tree structure. You receive an error if the tree structure already exists.

The Copy method can only be used with a closed tree structure, it cannot be used on an open tree structure. Before you use the Copy method, you must explicitly close any open tree structure objects (with the Close method.) You receive an error if there are any open tree structures.

To access the new tree structure, you must use the Open method.

Returns

A number; 0 if the method completed successfully.

Example

```
&MyTreeStruct = &MySession.GetTreeStructure();
&RetValue = &MyTreeStruct.Copy(PERSONAL_DATA, PERSONAL_DATA2);
```

Create

Syntax

Create*(StructId, Type)*

Description

The Create method creates a tree structure with the name *StructId*, and of the type *Type*. Both parameters take a string value.

Note. If the tree structure already exists, it is overwritten with the newly created tree structure.

The Create method can be used only with a closed tree structure, it cannot be used on an open tree structure. Before you use the Create method, you must explicitly close any open tree structure objects (with the Close method.) You receive an error if there are any open tree structures.

The new tree structure must be saved (using the Save method) for this structure to be saved to the database.

After you create a new tree structure, you can open it with the Open method.

The values for *Type* are:

- "D": create a detail tree structure.
- "S": create a summary tree structure.

Currently, only the value of "D" is supported for the *Type* parameter. You can specify an empty string for *Type*. If you need to create a summary structure follow the example below:

```
&MYSTRUCT=&SESSION.GETSTRUCTURE();
IF ALL( &MYSTRUCT) THEN
   &RETVALUE=&MYSTRUCT.CREATE('DEPARTMENT_SUM',' ');
     IF NONE(&RETVALUE) THEN
        &MYSTRUCT.TYPE = 'S';
     END-IF;
   &RETVALUE=&MYSTRUCT.SAVE();
   &RETVALUE=&MYSTRUCT.CLOSE();
END-IF;
```


Delete

Syntax

`Delete`_`(StructId)`_

Description

The Delete method deletes the tree structure specified by *StructId* from the database. The Delete method can be used only with a tree structure identifier, it cannot be used on a fully instantiated, open tree structure. Before you use the Delete method, you must explicitly close any open tree structure objects (with the Close method.) You receive an error if there are any open tree structures.

The tree structure specified by *StructId* does not have to match the tree structure identifier executing the method.

Returns

A number: 0 if the method completed successfully.

Open

Syntax

`Open`_`(StructId,Update)`_

Description

The Open method opens the tree structure specified by *StructId.StructId* takes a string value. You cannot read or set any properties of a tree structure until after you open it.

When you open a tree structure, you can specify whether you want to open it for update, or in read-only mode, with the *Update* parameter. *Update* takes a Boolean value. Values for *Update* are:

- "True": open tree structure in update mode (read-write).
- "False": open tree structure in read-only mode.

If you try updating or writing to a tree structure opened in read-only mode, your changes are ignored.

Returns

A number: 0 if the method completes successfully.

Example

```
&MyTreeStruct = &MySession.GetTreeStructure();
&REtValue = &MyTreeStruct.Open("ACCESS_GROUP", True);
```

Rename

Syntax

Rename*(FromStructId,ToStructId)*

Description

The Rename method renames the tree structure specified with *FromStructId* to the new name specified by *ToStructId.* Both parameters take a string value. The tree structure specified by *ToStructId* must be a *new* tree structure. You receive an error if the tree structure already exists. The change is automatically reflected in the database.

The Rename method can only be used with a closed tree structure, it cannot be used on an open tree structure. Before you use the Rename method, you must explicitly close any open tree structure objects (with the Close method.) You receive an error if there are any open tree structures.

Returns

A number: 0 if the method completes successfully.

Save

Syntax

Save()

Description

The Save method writes any changes to the tree structure executing the method to the database.

The Save method can be used only on an open tree structure, not on a closed tree structure. This means you must have opened the tree structure with the Open method before you can save it.

The tree structure object remains open after executing Save. You must execute the Close method on the object before it is closed and the memory freed.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a Commit.

Returns

A number: 0 if the method completes successfully.

Tree Structure Class Properties

In this section, we discuss the Tree Structure class properties. The properties are discussed in alphabetical order.

Description

Description

This property returns the description for the tree structure, as a string. The description is the text that displays in list boxes or reports.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

Example

```
&MYTREESTRUCT.Description = "Department Security Chart";
```

DetailComponent

Description

This property returns the name of the detail component for the tree structure as a string.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

DetailField

Description

This property returns the name of the detail field for the tree structure, as a string. This property is valid only if you define a detail tree structure. If you create a node-oriented tree structure, this property is not required. This field must exist on the specified DetailRecord.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

Example

The following example sets the DetailPage, DetailRecord, and DetailField properties for a tree structure. Note that the names of the page, record, and field are all capitalized: this is required for all PeopleSoft page, record, and field names.

```
&MYTREESTRUCT.DetailPage = "BUS_UNIT_TREE_INV";
&MYTREESTRUCT.DetailRecord = "BUS_UNIT_TBL_IN";
&MYTREESTRUCT.DetailField = "BUSINESS_UNIT";
```

See Also

Chapter 43, "Tree Classes," DetailRecord, page 2410

DetailMenu

Description

This property returns the name of the detail menu for the tree structure, as a string. If the page you use for detail values (DetailPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the DetailMenu and DetailMenuBar properties.

This property is valid only if you define a detail tree structure. If you create a node-oriented tree structure, this property isn't required.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

DetailMenuBar

Description

This property returns the name of the detail menu bar for the tree structure, as a string. If the page you use for detail values (DetailPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the DetailMenu and DetailMenuBar properties.

This property is valid only if you're define a detail tree structure. If you create a node-oriented tree structure, this property isn't required.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

DetailMenuItem

Description

This property returns the name of the detail menu item for the tree structure as a string.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

DetailMultiNavigate

Description

This property sets or returns a Boolean value indicating whether the detail mulit-navigation tree structure navigation option has been selected or not: True, the Detail Multi-Navigation check box has been selected, False otherwise.

This property is read-write.

See Also

Chapter 43, "Tree Classes," NodeMultiNavigate, page 2416 and *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager*, "Setting Multinavigation Paths"

DetailPage

Description

This property returns the name of the detail page for the tree structure as a string. If the page you use for detail values (DetailPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the DetailMenu and DetailMenuBar properties.

This property is valid only if you're defining a detail tree structure. If you're creating a node-oriented tree structure, this property isn't required.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Example

The following example sets the DetailPage, DetailRecord, and DetailField properties for a tree structure. Note that the names of the page, record, and field are all capitalized: this is required for all PeopleSoft page, record, and field names.

```
&MYTREESTRUCT.DetailPage = "BUS_UNIT_TREE_INV";
&MYTREESTRUCT.DetailRecord = "BUS_UNIT_TBL_IN";
&MYTREESTRUCT.DetailField = "BUSINESS_UNIT";
```

DetailRecord

Description

This property returns the name of the detail record for the tree structure as a string. This property is valid only if you define a detail tree structure. If you create a node-oriented tree structure, this property isn't required.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Example

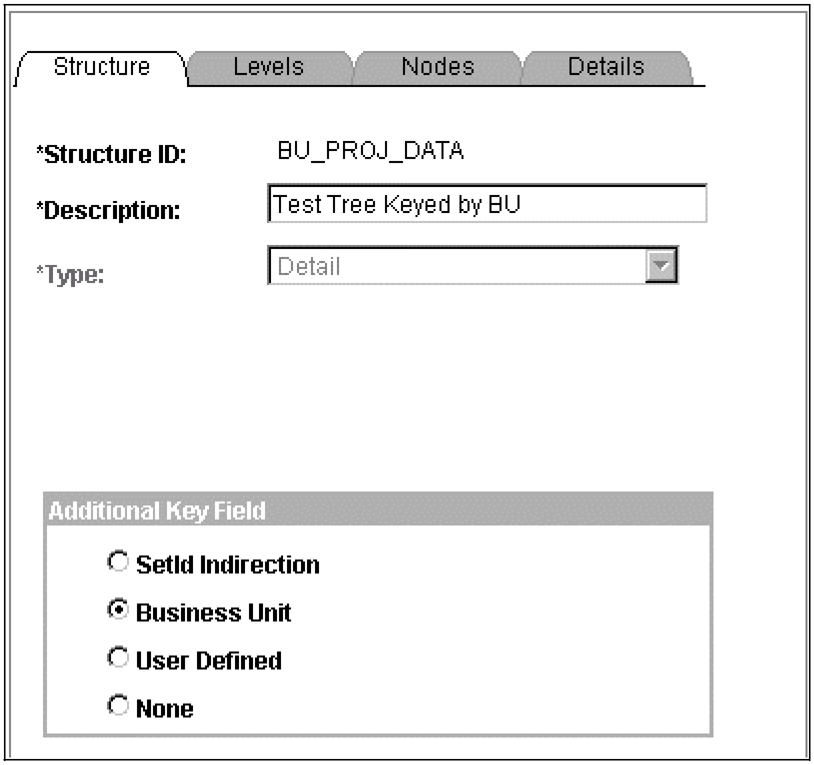
The following example sets the DetailPage, DetailRecord, and DetailField properties for a tree structure. Note that the names of the page, record, and field are all capitalized: this is required for all PeopleSoft page, record, and field names.

```
&MYTREESTRUCT.DetailPage = "BUS_UNIT_TREE_INV";
&MYTREESTRUCT.DetailRecord = "BUS_UNIT_TBL_IN";
&MYTREESTRUCT.DetailField = "BUSINESS_UNIT";
```


IndirectionMethod

Description

When you set up a tree structure, you can set the indirection at the bottom of the Structure tab, with the radio buttons in the Additional Key Field.



Structure tab for Tree Structure setup

In the Tree Classes API, you set these same values with the IndirectionMethod property. The values are:

- "S": SetID Indirection.
- "B": Business Unit.
- "U": User Defined Node Key.
- "N": None.

Both the *User Defined Node Key* and the *Business Unit* are the key field set with the NodeRecord property for this tree structure. If *Business Unit* is set, the key field set with the NodeRecord property is a business unit.

If you are creating a new tree structure, and you do not explicitly set this property, None ("N") is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

See Also

Chapter 43, "Tree Classes," IndirectionMethod, page 2411 and Chapter 43, "Tree Classes," NodeRecord, page 2417

KeyName

Description

This property returns the name of the tree structure as a string.

This property can be used with a closed tree structure.

This property is read-only.

LevelComponent

Description

This property returns the name of the level component for the tree structure as a string.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelMenu

Description

This property returns the name of the level menu for the tree structure as a string. If the page you use for level values (LevelPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the LevelMenu and LevelMenuBar properties.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelMenuBar

Description

This property returns the name of the level menu bar for the tree structure as a string. If the page you use for level values (LevelPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the LevelMenu and LevelMenuBar properties.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelMenuItem

Description

This property returns the name of the level menu item for the tree structure as a string.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelPage

Description

This property returns the name of the level page for the tree structure object as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_LEVEL is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelRecord

Description

This property returns the name of the level record for the tree structure object as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_LEVEL_TBL is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Name

Description

This property returns the name of the tree structure as a string. This is also called the *structure ID.* Use this property to set the name (structure ID) of a tree structure. This property is required if you are creating a new tree structure.

Note. Do not use this property to change the name of an existing tree structure. The change will not be reflected in the database. You must use the Rename method to change the name of an existing tree structure.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Example

```
&Name = &MYTREESTRUCT.Name;
```

See Also

Chapter 43, "Tree Classes," Rename, page 2406

NodeComponent

Description

This property returns the name of the node component for the tree structure as a string.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

NodeField

Description

This returns the name of the field used for storing information about nodes for the tree structure as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_NODE is the default value, and is automatically set. This field must exist on the specified NodeRecord.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Example

```
&MYTREESTRUCT.NodeField = "DEPT_ID";
```

NodeMenu

Description

This property returns the name of the node menu for the tree structure as a string. If the page you use for node values (NodePage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the NodeMenu and NodeMenuBar properties.

If you create a new tree structure, and you do not explicitly set this property, TREE_MANAGER is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

NodeMenuBar

Description

This property returns the name of the node menu bar for the tree structure as a string. If the page you use for node values (NodePage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the NodeMenu and NodeMenuBar properties.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

NodeMenuItem

Description

This property returns the name of the node menu item for the tree structure as a string.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

NodeMultiNavigate

Description

This property sets or returns a Boolean value indicating whether the node mulit-navigation tree structure navigation option has been selected: True, the Node Multi-Navigation check box has been selected, False otherwise.

This property is read-write.

See Also

Chapter 43, "Tree Classes," DetailMultiNavigate, page 2409 and *Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Tree Manager*, "Setting Multinavigation Paths"

NodePage

Description

This returns the name of the page used for storing information about nodes for the tree structure as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_NODE is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

NodeRecord

Description

This returns the name of the record used for storing information about nodes for the tree structure as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_NODE_TBL is the default value, and is automatically set.

If the IndirectionMethod property of this tree structure has been set to "U" or "B", use this property to specify the record that the NodeUserKeyField property will use.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

See Also

Chapter 43, "Tree Classes," IndirectionMethod, page 2411

NodeUserKeyField

Description

Use this property to set the key field from the record specified with NodeRecord for use with the IndirectionMethod. This property is identical to the UserKeyValue property for a tree.

This property is used only when the IndirectionMethod property for the tree structure has been set to "U" or "B". Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

See Also

Chapter 43, "Tree Classes," IndirectionMethod, page 2411 and Chapter 43, "Tree Classes," UserKeyValue, page 2402

SummarySetId

Description

This property returns the SetID for the detail tree whose detail values the summary tree structure as a string summarizes. This property is valid only for Summary tree structures. Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

SummaryLevelNumber

Description

This property returns the level number for the detail tree whose detail values the summary tree structure summarizes. The level number indicates the level in the detail tree whose nodes the summary tree will use as detail values. The top level in the detail tree is 1, the next is level two, and so on. Valid values are numbers between 1 and 99.

This property is only valid for Summary tree structures. Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

SummaryTreeName

Description

This property returns the name of the detail tree whose detail values the summary tree structure summarizes, as a string. This property is valid only for Summary tree structures. Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

SummaryUserKeyValue

Description

This property returns the name of the tree that the node records the tree structure is summarized to, as a string. This property is valid only for Summary tree structures. This property is used only when the IndirectionMethod property for the tree structure has been set to "U" for *User Defined Keys* or "B" for *Business Unit.* Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

See Also

Chapter 43, "Tree Classes," IndirectionMethod, page 2411

Type

Description

This property returns the type of the tree structure as a string, whether it is a detail or summary tree structure. The values are:

- "D": detail tree structure.
- "S": summary tree structure.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Traverse Tree Hierarchy Example

The following is an example to transverse a tree hierarchy and write the information to a file.

```
Local ApiObject &TreeObj, &RootNodeObj;
Local number &Res;
Local string &sSetId, &sSetCntrlValue, &sTreeName, &sBranchName;
Local File &MyFile;

Function ProcessNodeChildren(&ParentNodeObj As ApiObject)

   Local ApiObject &ChildNodeObj, &ChildLeafObj;
   Local boolean &First;

   If &ParentNodeObj.HasChildren Then

      &ChildLeafObj = &ParentNodeObj.FirstChildLeaf;
      While All(&ChildLeafObj)
         &MyFile.WriteLine("Process Leaf Range From : " | &ChildLeafObj.Range⇒
From);
         &ChildLeafObj = &ChildLeafObj.NextSib;
      End-While;

      If &ParentNodeObj.HasChildNodes Then
         &First = True;
         &ChildNodeObj = &ParentNodeObj.FirstChildNode;
         While &First Or
               &ChildNodeObj.HasNextSib

            If &First Then
               &First = False;
            Else
               &ChildNodeObj = &ChildNodeObj.NextSib;
            End-If;
            &MyFile.WriteLine("Process Node : " | &ChildNodeObj.Name);
            ProcessNodeChildren(&ChildNodeObj);

         End-While;
      End-If;
   End-If;
End-Function;

&TreeObj = %Session.GetTree();

If None(&TreeObj) then
   Error("Cannot get a tree");
End-if;

&sSetId = "";
&sSetCntrlValue = "";
&sTreeName = "QE_JOBCODES";
&TreeEffDt = Date("1999-01-01");
&sBranchName = "";

&Res = &TreeObj.OPEN(&sSetId, &sSetCntrlValue, &sTreeName, &TreeEffDt, &sBranch⇒
Name, False);
If All(&Res) Then
   Error("Cannot open a tree");
End-If;

&MyFile = GetFile("TreeDump.txt", "A");
&RootNodeObj = &TreeObj.FindRoot();

&MyFile.WriteLine("Process Root Node : " | &RootNodeObj.Name);

ProcessNodeChildren(&RootNodeObj);
&MyFile.Close();
```

Chapter 44

Universal Queue Classes

This chapter provides an overview of the universal queue classes and discusses the following topics:

- MCFFactory class hierarchy
- Scope of universal queue classes
- Data types of universal queue classes
- How to import universal queue classes
- How to create universal queue class objects
- Universal queue classes reference

Understanding Universal Queue Classes

The universal queue classes provide the means by which applications can inspect objects that are processed by the queue server, such as tasks, agents, and queues. The classes also enable tasks to reenter the queue with their original task ID, as well as keep task times based on its original entry into the system.

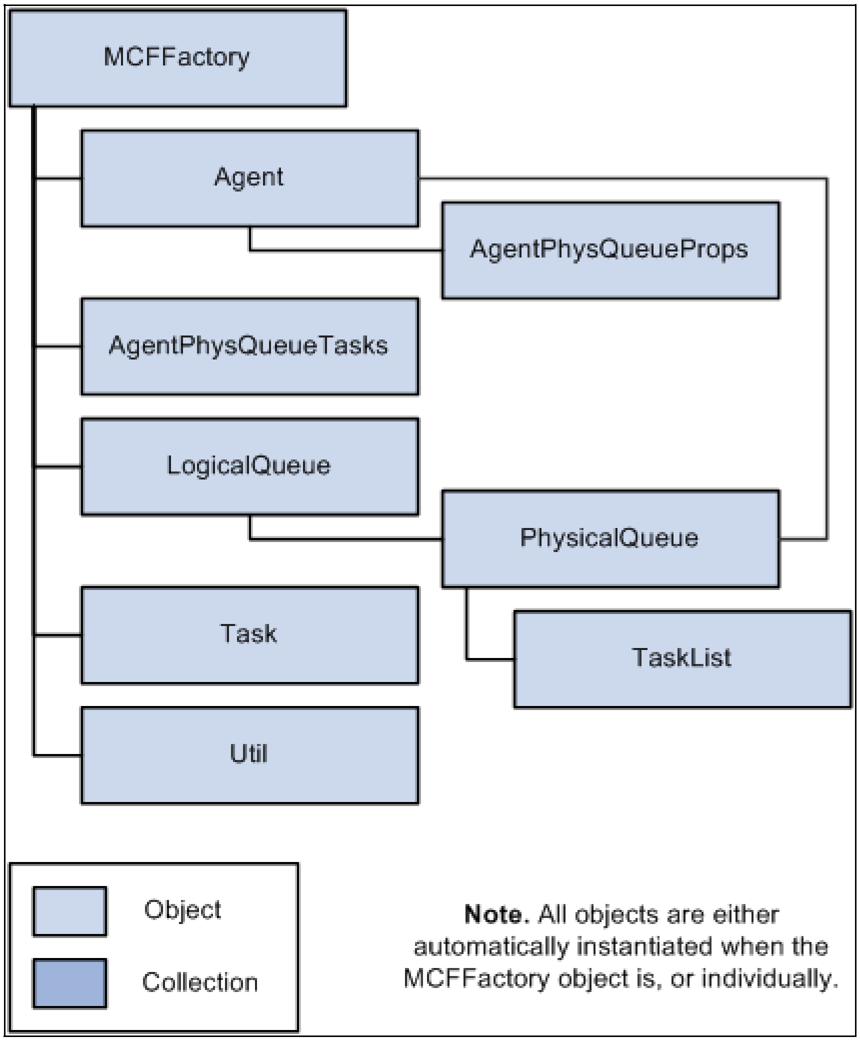
All of the classes, and most of the properties and methods that make up the universal queue classes have a GUI representation in the PeopleSoft Multi-Channel Framework. This document assumes that the reader is familiar with PeopleSoft Multi-Channel Framework.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft MultiChannel Framework, "Understanding PeopleSoft MultiChannel Framework"

MCFFactory Class Hierarchy

The MCFFactory class is the root class from which the other classes are derived. Instantiate an object in this class, then traverse down the properties to inspect other individual objects, such as tasks, agents or queues. You can also instantiate individual MCFFactory objects directly and inspect their properties.



MCFFactory class hierarchy

Scope of the Universal Queue Classes

The universal queue classes can be instantiated only from PeopleCode.

The universal queue classes can be called from a component, an internet script, or an Application Engine program.

Universal queue classes can be of Local, Global, or Component scope.

Data Types of the Universal Queue Classes

Every universal queue class is its own data type, that is, Agent objects are declared as data type Agent, LogicalQueue objects are declared as type LogicalQueue, and so on.

The following are the data types of the universal queue classes:

- Agent
- AgentPhysQueueProps
- AgentPhysQueueTasks
- Broadcast
- LogicalQueue
- MCFFactory
- PhysicalQueue
- Task
- TaskList
- Util

How to Import Universal Queue Classes

The universal queue classes are *not* built-in classes, like rowset, field, record, and so on. They are application classes. You must import the universal queue application package before you can use the classes in your PeopleCode program.

An import statement names either all the classes in a package or one particular application class. For importing the universal queue classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_MCF_UQAPI contains the following sub-packages:

- Agent
- AgentPhysQueueProps
- AgentPhysQueueTasks
- Broadcast
- LogicalQueue
- MCFFactory
- PhysicalQueue
- Task
- TaskList
- Util

The import statements you should use are as follows:


```
import PT_MCF_UQAPI:Agent:*;
import PT_MCF_UQAPI:AgentPhysQueueProps:*;
import PT_MCF_UQAPI:AgentPhysQueueTasks:*;
import PT_MCF_UQAPI:Broadcast:*;
import PT_MCF_UQAPI:LogicalQueue:*;
import PT_MCF_UQAPI:MCFFactory:*;
import PT_MCF_UQAPI:PhysicalQueue:*;
import PT_MCF_UQAPI:Task:*;
import PT_MCF_UQAPI:TaskList:*;
import PT_MCF_UQAPI:Util:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available.

See Also

Chapter 6, "Application Classes," page 189

How to Create a Universal Queue Object

After you've imported the universal queue classes, you instantiate an object of one of those classes using the constructor for the class and the *Create* function.

The following example creates a new instance of the MCFFactory class, as the variable &MyMCFFactory, with local scope:

```
Local MCFFactory &myMCFFactory  = Create MCFFactory ();
```

See Also

Chapter 44, "Universal Queue Classes," Universal Queue Classes Constructors, page 2425

Universal Queue Classes Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," DeQueue

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," EnQueue

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Forward

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," InitChat

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," MCFBroadcast

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," NotifyQ

Universal Queue Classes Constructors

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the universal queue Classes.

Agent

Syntax

`Agent`*`(AgentID)`*

Description

Use the Agent constructor to instantiate an Agent object.

All objects encapsulated by this object are created when the constructor is called.

Note. When an MCFFactory object is created, Agent objects are automatically created. In this case, the number of tasks in the list is limited by the *MAX_TASKLIST_ITEMS* parameter used when creating the MCFFactory object.

Parameters

Parameter	***Description***
AgentID	Specify a valid AgentID as a string.

Returns

An Agent object.

Example

```
Create Agent(&AgentID);
```

See Also

Chapter 44, "Universal Queue Classes," Agent Class, page 2433

AgentPhysQueueProps

Syntax

`AgentPhysQueueProps``(AgentID, PhysicalQueueID)`

Description

Use the AgentPhysQueueProps constructor to instantiate an agent physical queue properties object.

Parameters

Parameter	***Description***
AgentId	Specify as valid AgentId as a string.
PyhscialQueueId	Specify a PhyscialQueueId as a string.

Returns

An agent physical queue properties object.

See Also

Chapter 44, "Universal Queue Classes," AgentPhysQueueProps Class, page 2439

AgentPhysQueueTasks

Syntax

`AgentPhysQueueTasks``(AgentID, PhysicalQueueID)`

Description

Use the AgentPhysQueueTasks constructor to instantiate an agent physical queue task object.

Parameters

Parameter	Description
AgentID	Specify an agent ID as a string.
PhysicalQueueID	Specify a PhysicalQueueID as a string.

Returns

An agent physical queue task object.

See Also

Chapter 44, "Universal Queue Classes," AgentPhysQueueTasks Class, page 2441

Broadcast

Syntax

```
Broadcast()
```

Description

Use the Broadcast constructor to instantiate a Broadcast object.

Parameters

None.

Returns

A Broadcast object.

See Also

Chapter 44, "Universal Queue Classes," Broadcast Class, page 2443

LogicalQueue

Syntax

`LogicalQueue`*`(LogicalQueueID)`*

Description

Use the LogicalQueue constructor to instantiate a LogicalQueue object.

Parameters

Parameter	***Description***
LogicalQueueID	Specify the ID of a logical queue as a string.

Returns

A LogicalQueue object.

Example

```
Local LogicalQueue &MyLogQueue = Create LogicalQueue(&LogicalQueueID);
```

See Also

Chapter 44, "Universal Queue Classes," LogicalQueue Class, page 2445

MCFFactory

Syntax

`MCFFactory``([`*`MAX_TASKLIST_ITEMS`*`])`

Description

Use the MCFFactory constructor to instantiate an MCFFactory object.

Parameters

Parameter	*Description*
MAX_TASKLIST_ITEMS	Specify the maximum number of tasks loaded on a TaskList.

Returns

An MCFFactory object.

Example

```
Local MCFFactory &myMCFFactory = Create MCFFactory(MAX_TASKLIST_ITEMS);
```

See Also

Chapter 44, "Universal Queue Classes," MCFFactory Class, page 2446

PhysicalQueue

Syntax

PhysicalQueue*(PhysicalQueueID)*

Description

Use the PhysicalQueue constructor to instantiate a PhysicalQueue object. All objects encapsulated by this object are created when the constructor is called.

Note. When an MCFFactory object is created, the PhysicalQueue objects are automatically created. In this case, the number of tasks in the list is limited by the *MAX_TASKLIST_ITEMS* parameter used when creating the MCFFactory object.

Parameters

Parameter	*Description*
PhysicalQueueID	Specify a physical queue ID as a string.

Returns

A PhysicalQueue object.

Example

```
Local PhysicalQueue &myPhysicalQueue = Create PhysicalQueue(&PhysicalQueueID);
```

See Also

Chapter 44, "Universal Queue Classes," PhysicalQueue Class, page 2447

Task

Syntax

Task*(TaskNumber)*

Description

Use the Task constructor to instantiate a task object.

Only persistent tasks (email and generic tasks) are available for introspection. Each persistent task goes through a lifecycle that is recorded by the following statuses:

State	*Indicated Action*	*Status of Task Object*
Enqueued by the universal queue server	Enqueued	ENQ
Universal queue assigns to physical queue	Assigned	ASGN
Agent accepts task	Accepted	ACPT
Agent completes task	Done	DONE
Task is not accepted within the overflow (action) time period	Overflowed	OVFL
Agent does not complete task or task not assigned within the escalation (complete) time period	Escalated	ESCL
Inderminate Task (task not found in system)	Indeterminate	IND

Parameters

Parameter	Description
TaskNumber	Specify a task number.

Returns

A task object if the task exists, or an error message if the task number is not in the expected format. If the task number is properly formatted, but the task is not found, the task object is returned with a task status of IND.

Example

```
Local string &tasknumber, &status;

Local Task &myTask = Create Task(&tasknumber);

&status = &myTask.Status;
```

See Also

Chapter 44, "Universal Queue Classes," Task Class, page 2453

TaskList

Syntax

TaskList(*Status*,*TaskType*,*PhysicalQueueID*, *AgentID*)

Description

Use the TaskList constructor to instantiate a task list.

If the queue is not provided, a list of tasks for all physical queues for that *TaskType* and *Status* is returned. To get all tasks on all queues do not specify any of the optional parameters.

Parameters

Parameter	Description
Status	Specify a task status, as an uppercase string. If you do not specify a valid status, an empty tasklist is returned. Valid values are:

Value	***Description***
ACPT	Accepted
ASGN	Assigned
ENQ	Enqueued
ESCL	Escalated
OVFL	Overflowed

Parameter	***Description***
TaskType	Specify a task type as a lowercase string. Valid values are:

Value	***Description***
chat	Chat tasks.
email	Email tasks.
generic	Generic type tasks.
voice	Voice type tasks.

Parameter	***Description***
PhysicalQueueID	Specify a physical queue ID as a string.
AgentID	Specify an agent ID as a string.

Returns

A task list object.

Example

```
Local string &sStatus;

/* get all enqueued task on the UQ server*/

&sStatus = "ENQ";
Local TaskList &myTaskList = Create TaskList (&sStatus, "", "", "");
```

See Also

Chapter 44, "Universal Queue Classes," TaskList Class, page 2459

Util

Syntax

Util()

Description

Use the Util constructor to instantiate a util object.

Parameters

None.

Returns

A Util object.

Example

```
Create Util();
```

```
Local &myUtil = create Util();
```

See Also

Chapter 44, "Universal Queue Classes," Util Class, page 2462

Agent Class

Use the agent class to refresh or view the properties, such as the agent ID or buddy list associated with a particular agent.

See Also

PeopleTools 8.51 PeopleBook: PeopleSoft MultiChannel Framework, "Configuring PeopleSoft MCF Agents"

Agent Methods

In this section, we discuss the agent class methods. The methods are discussed in alphabetical order.

Delete

Syntax

Delete()

Description

Use the Delete method to delete an agent from the database. You cannot delete an agent that has still has open tasks.

Parameters

None.

Returns

A number. The values are:

Value	***Description***
0	Agent deleted successfully.
1	Invalid agent ID.
2	Agent has accepted task and cannot be deleted.
3	Other SQL error in executing deletion. Possibly there are errors in the child tables.

Example

```
import PT_MCF_UQAPI:*;


   &testAgent = create PT_MCF_UQAPI:Agent("QEMGR");

   /* Delete an UQ Agent definition */
   &retCode = &testAgent.Delete();
```

Refresh

Syntax

`Refresh()`

Description

Use the Refresh method to refresh the properties of the agent.

An agent on a queue is processing tasks as they are assigned to them. Use this method to see realtime statistics on how much load an agent has, or possibly before running any metrics.

If you're interested in how tasklists have changed, use the RefreshQTaskList method instead.

Parameters

None.

Returns

None.

Example

The following refreshes all the properties of the agent:

```
&MyAgent.Refresh();
```

See Also

Chapter 44, "Universal Queue Classes," RefreshQTaskList, page 2435

RefreshQTaskList

Syntax

RefreshQTaskList*(PhysQ)*

Description

Use the RefreshQTaskList method to refresh the task list properties for the specified physical queue.

You might use this method if you're interested in seeing the realtime statistics for a particular tasklist. If you are only interested in an agent's work, use the Refresh method.

Parameters

Parameter	***Description***
PhysQ	Specify a physical queue ID as a string.

Returns

None.

Example

The following refreshes all task lists on the first physical queue of the agent:

```
&MyAgent.RefreshQTaskList(&myAgent.PhysicalQueueID [1]);
```

See Also

Chapter 44, "Universal Queue Classes," Refresh, page 2435

Agent Properties

In this section, we discuss the agent class properties. The properties are discussed in alphabetical order.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

AgentProps

Description

This property returns the agent properties associated with this agent as an array of AgentPhysQueueProps objects.

This property is read-only.

Example

The following example returns the workload for physical queue SALES of agent properties object:

```
Local number &i = 1;

While &i < &myAgent.TotalPhysicalQueues
   If &myAgent.AgentProps[&i].PhysicalQueueID = "SALES" Then
      &AgentWorkLoad = &myAgent.AgentProps[&i].WorkLoad
      break;
   End-If;
End-While;
```

See Also

Chapter 44, "Universal Queue Classes," AgentPhysQueueProps Class, page 2439

AgentTasks

Description

This property returns the agent tasks associated with this agent as an array of AgentPhysQueueTasks objects.

This property is read-only.

Example

The following example returns the task number for the second task on physical queue SALES in the accepted task list:

```
Local Number &i = 1;

While &i < &myAgent.TotalPhysicalQueues
   If &myAgent.AgentProps[&i].PhysicalQueueID = "SALES" then
      &Tasknumber = &myAgent.AgentsTasks [&i].AcceptedTaskList.Task[2].TaskNumber
      break;
   End-If;
End-While;
```

See Also

Chapter 44, "Universal Queue Classes," AgentPhysQueueTasks Class, page 2441

Buddy

Description

This property returns a list of buddies for this agent, as an array of string.

This property is read-only.

Example

The following example returns the total number of buddies on the list:

```
&BuddyTotal = &myAgent.Buddy.Len;
```

Language

Description

This property returns the list of languages associated with the agent as an array of string.

This property is read-only.

Example

The following code example returns the first language code in the list of languages associated with the agent:

```
&Language = &myAgent.Language[1];
```

Name

Description

This property returns the name of the agent as a string.

This property is read-only.

NickName

Description

This property returns the nick name of the agent as a string.

This property

PhysicalQueueID

Description

This property returns the physical queues to which this agent is assigned as an array of string.

This property is read-only.

Example

The following example returns the first physical queue of agent.

```
&PhysicalQueueID = &myAgent.PhysicalQueueID[1];
```

See Also

Chapter 44, "Universal Queue Classes," PhysicalQueue Class, page 2447

TotalPhysicalQueues

Description

This property returns the total number of physical queues to which this agent is assigned as a number.

This property is read-only.

Example

```
&TotalPhysicalQueues = &myAgent.TotalPhysicalQueues;
```

See Also

Chapter 44, "Universal Queue Classes," PhysicalQueue Class, page 2447

AgentPhysQueueProps Class

Use the AgentPhysQueueProps class to view the properties of the physical queue assigned to an agent.

AgentPhysQueueProps Properties

In this section, we discuss the agent physical queue properties class properties. The properties are discussed in alphabetical order.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

PhysicalQueueID

Description

This property returns the physical queue ID as a string.

This property is read-only.

Example

The following code example returns the physical queue ID for the physical queue of agent.

```
&PhyscalQueueID = &myAgentPhysQueueProps.PhysicalQueueID;
```

See Also

Chapter 44, "Universal Queue Classes," PhysicalQueue Class, page 2447

SkillLevel

Description

This property returns the skill level as a number.

This property is read-only.

Example

The following code example returns the skill level on physical queue.

```
&SkillLevel = &myAgentPhysQueueProps.SkillLevel;
```

WorkLoad

Description

This property returns the maximum work load that can be assigned to the agent, as an integer.

This property is read-only.

AgentPhysQueueTasks Class

Use the AgentPhysQueueTasks class to refresh or view the tasks associated with the physical queue assigned to the agent.

AgentPhysQueueTasks Method

In this section, we discuss the agent physical queue tasks class method Refresh.

Refresh

Syntax

`Refresh`*`(TaskList)`*

Description

Use the Refresh method to refresh the specified task list.

Parameters

Parameter	***Description***
TaskList	Specify the task list you want refereshed. You must specify a task list for this parameter.

Returns

None.

Example

The following code example refresehs the accepted task list of the physical queue (&MyAgentPhysQueueTasks) for the agent.

```
&myAgentPhysQueueTasks.Refresh(&myAgentPhysQueueTasks.AcceptedTaskList);
```

See Also

Chapter 44, "Universal Queue Classes," TaskList Class, page 2459

AgentPhysQueueTasks Properties

In this section, we discuss the agent physical queue tasks class properties. The properties are discussed in alphabetical order.

AcceptedTaskList

Description

This property returns the accepted task list as a task list object.

This property is read-only.

Example

The following code example returns the task number of the first accepted task:

```
&TaskNumber = &myAgentPhysQueueTasks.AcceptedTaskList.Task[1].TaskNumber;
```

AssignedTaskList

Description

This property returns the assigned task list as a task list object.

This property is read-only.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

PhysicalQueueID

Description

This property returns the physical queue ID as a string.

This property is read-only.

Example

The following code example returns the physical queue ID for the physical queue of agent.

```
&PhyscalQueueID = &myAgentPhysQueueTasks.PhysicalQueueID;
```

See Also

Chapter 44, "Universal Queue Classes," PhysicalQueue Class, page 2447

Broadcast Class

The Broadcast class has a single method, Broadcast. Use the Broadcast method to broadcast a notification message. You can specify whether to send the message to agents, to a queue, or even system wide.

Broadcast Class Method

The following is the description of the Broadcast class method Broadcast.

Broadcast

Syntax

`Broadcast``(ClusterID,QueueID,ChannelID, AgentState,AgentPresence,Message, MessageSetNumber,MessageNumber,DefaultMessage,SecurityLevel,ImportanceLevel, SenderId,NameValueString)`

Description

Use the Broadcast function to broadcast a notification message. You can specify whether to send the message to agents, to a queue, or even system wide.

Parameters

Parameter	***Description***
ClusterID	Specify the name of the cluster that you want to broadcast the message to, such as, RENCLSTR_001, as a string.
QueueID	Specify the name of the physical or logical queue that you want to broadcast the message to, such as, SALES, as a string.
ChannelID	Specify the name of the channel, or task, for the broadcast, such as Email, Chat, Voice or Generic, as a string.
AgentState	Specify the state of the agents you want to broadcast the message to, such as Available, as a string.
AgentPresence	Specify the presence of the agents you want to broadcast the message to, such as Active, as a string.
Message	Specify the text of the message you want to broadcast, as a string.
MessageSetNumber	Specify the message set number of a message from the message catalog if you want to broadcast a message from the message catalog. You must also specify values for the *MessageNumber* and *DefaultMessageText* parameters if you want to broadcast this type of message. Specify the message set number as a number.
MessageNumber	Specify the message number of a message from the message catalog if you want to broadcast a message from the message catalog. You must also specify values for the *MessageSetNumber* and *DefaultMessageText* parameters if you want to broadcast this type of message. Specify the message number as a number.
DefaultMessageText	Specify the text to be used if the specified message catalog message isn't found. Use the *MessageSetNumber* and *MessageNumber* parameters to specify the catalog message. Specify the default message text as a string.
SercurityLevel	Specify the security level for the broadcast message, as a string.

Parameter	***Description***
ImportanceLevel	Specify the importance level of the broadcast message, as a string.
SenderID	Specify the ID of the sender of the broadcast message, as a string.
NameValueString	Specify a string containing name-value pairs specific to your application.

Returns

None.

Example

The following example would broadcast a message to a specific logical queue:

```
import PT_MCF_UQAPI:Broadcast:*;
Local PT_MCF_UQAPI:Broadcast &BC;

&BC.Broadcast("", "SALES", "", "", "Best of Luck!", "", "", "Default Message",⇒
 "PRIV1", "URGENT", "Admin", "EffDate, 2005-10-25:12:00:45");
```

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," MCFBroadcast

LogicalQueue Class

A logical queue is an application level queue that receives work requests (tasks) relating to an application area, such as chat requests regarding sales information, and routes them to agents capable of handling the work. Use the LogicalQueue class to view the properties for a logical queue.

LogicalQueue Properties

In this section, we discuss the logical queue class properties. The properties are discussed in alphabetical order.

LogicalQueueID

Description

This property returns the logical queue ID as a string.

This property is read-only.

PhysicalQueueID

Description

This property returns all the physical queues associated with this logical queue as an array of PhysicalQueue objects.

This property is read-only.

Example

The following code example returns the physical queue ID for the fourth physical queue.

```
&PhyscalQueueID = &myLogicalQueue.PhysicalQueue[4].PhyscalQueueID;
```

MCFFactory Class

The MCFFactory class is the base class of the universal queue classes. Instantiating an object in this class automatically instantiates all the associated universal queue classes. The MCFFactory contains all the logical queues operating on the MCF universal queue. The logical queues are comprised of physical queues. Agents log onto these physical queues. Tasks are queued to physical queues, and later, assigned and accepted by agents. If agents aren't available on a physical queue to accept the tasks on that queue, the tasks are either escalated or put into overflow.

MCFFactory Property

In this section, we discuss the MCFFactory class property LogicalQueue.

LogicalQueue

Description

This property returns all the logical queues with which this MCFFactory object is associated, as an array of LogicalQueue objects.

This property is read-only.

Example

The following code example returns the physical queue ID for the first physical queue on the logical queue, as well as the total of overflowed taskes on the first physical queue on the logical queue.

```
Local number &TotalTasks;

Local MCFFactory &myMCFFactory  = Create MCFFactory ();

&PhyscalQueueID = &myMCFFactory.LogicalQueue [1].PhysicalQueue [1].PhysicalQueueID;

&TotalTasks  = &myMCFFactory.LogicalQueue [1].PhysicalQueue [1].OverflowedTask⇒
List.Total;
```

PhysicalQueue Class

Logical queues can be partitioned into physical queues for scalability. Use the PhysicalQueue class to refresh and view the properties of a physical queue.

PhysicalQueue Methods

In this section, we discuss the physical queue class methods. The methods are discussed in alphabetical order.

Refresh

Syntax

Refresh()

Description

Use the Refresh method to refresh all the objects in the physical queue.

Parameters

None.

Returns

None.

Example

```
&myPhysicalQueue.Refresh();
```

RefreshTaskList

Syntax

`RefreshTaskList`*`(TaskList)`*

Description

Use the RefreshTaskList method to refresh the specified task list.

Parameters

Parameter	***Description***
TaskList	Specify the task list you want refereshed. You must specify a task list for this parameter.

Returns

None.

Example

The following code example refreshes the accepted task list for the physical queue.

```
&myPhysicalQueue.RefreshTaskList(&myPhysicalQueue.AcceptedTaskList);
```


PhysicalQueue Properties

In this section, we discuss the physical queue class properties. The properties are discussed in alphabetical order.

AcceptedTaskList

Description

This property returns the accepted task list as a task list object.

This property is read-only.

Example

The following code example returns the task number of the first accepted task:

```
&TaskNumber = &myPhysicalQueue.AcceptedTaskList.Task[1].TaskNumber;
```

See Also

Chapter 44, "Universal Queue Classes," TaskList Class, page 2459

AssignedTaskList

Description

This property returns the assigned task list as a task list object.

This property is read-only.

See Also

Chapter 44, "Universal Queue Classes," TaskList Class, page 2459

Agent

Description

This property returns all the agents assigned to the physical queue as an array of agent objects.

This property is read-only.

Example

The following example returns the name of the third agent.

```
&AgentName = &MyPhysicalQueue.Agent[3].Name;
```

BrowserURL

Description

This property returns the URL of the browser as a string. The format is that of an absolute URL.

The URL for the REN server that serves this MCF cluster for external clients and for agent chat. The Browser URL may be different from the InternalURL, which should not have to go through any firewall, reverse proxy server or other outward-facing security barrier.

This property is read-only.

See Also

Chapter 44, "Universal Queue Classes," InternalURL, page 2451

EnqueuedTaskList

Description

This property returns the enqueued task list as a task list object.

This property is read-only.

See Also

Chapter 44, "Universal Queue Classes," TaskList Class, page 2459

EscalatedTaskList

Description

This property returns the escalated task list as a task list object.

This property is read-only.

See Also

Chapter 44, "Universal Queue Classes," TaskList Class, page 2459

InternalURL

Description

This property returns the internal URL as a string. The format is that of an absolute URL.

This property is read-only.

See Also

Chapter 44, "Universal Queue Classes," BrowserURL, page 2450

IsActive

Description

This property returns true if the physical queue is active, false otherwise.

This property is read-only.

Example

```
Local LogicalQueue &myLogicalQueue  = Create LogicalQueue (&LogicalQueueID);

For &I = 1 to &myLogicalQueue.PhysicalQueue.Len

&PhyscalQueueID = &myLogicalQueue.PhysicalQueue[&I].PhyscalQueueID;
&myPhysicalQueue = Create PhysicalQueue (&PhysicalQueueID);

   If &myPhysicalQueue.IsActive Then
      /* do work */
   Else
      /* return error */
   End-If;

End-For;
```

LogicalQueueID

Description

This property returns the logical queue ID associated with the physical queue as a string.

This property is read-only.

Example

```
&LogicalQueueID = &myPhysicalQueue.LogicalQueueID;
```

OverflowedTaskList

Description

This property returns the overflowed task list as a task list object.

This property is read-only.

See Also

Chapter 44, "Universal Queue Classes," TaskList Class, page 2459

PhysicalQueueID

Description

This property returns the physical queue ID of this physical queue.

This property is read-only.

RENURLID

Description

This property returns the ID of the REN server for the MCFFactory as a string.

This property is read-only.

TotalAgents

Description

This property returns the total number of agents belonging to this physical queue as a number. They may or may not be logged to the queue at that time.

This property is read-only.

Task Class

Use the task class to manipulate tasks or to view the properties of a specific task.

Task Methods

In this section, we discuss the task class methods. The methods are discussed in alphabetical order.

Close

Syntax

`Close`*`(Comment)`*

Description

Use the Close method to close the task. Only use this method with overflowed or escalated tasks.

Note. This method does *not* resubmit the task. Use the Enqueue method to resubmit the task, then use this method to close it.

Parameters

Parameter	*Description*
Comments	Specify any comments you want associated with this closed task. You can specify a null value `("")` for this parameter.

Returns

None.

See Also

Chapter 44, "Universal Queue Classes," Enqueue, page 2454

Enqueue

Syntax

Enqueue*(LogicalQueueID,AgentID,Timeout, ResponseTime,Cost,Priority,MinSkill)*

Description

Use the Enqueue method to add a task to the logical queue that the task was first assigned to. Every task enters the universal queue by being assigned to a logical queue.

Considerations Specifying Time

If no value (zero) is provided for the time parameters (such as *Timeout* or *ResponseTime)* the Timeout or ResponseTime properties from the task are used, respectively. This may cause the time to never occur as it creates a moving deadline. To avoid such a problem the time can be calculated for each task prior to enqueue as shown below.

Note. This logic is not included in the task's enqueue function, so as to allow you the choice for how you want your application to handle this situation.

```
&Utilobj = create Util();
&CurrentTask = create Task(&Task_Number);
&TimeDelta = &Utilobj.GetTimeDiff(&currenttime, &CurrentTask.EscalationTime) / 60;
&AgentAcceptTime = &Utilobj.GetTimeDiff(&CurrentTask.EnqueueTime, &Current⇒
Task.OverFlowTime) / 60;
    If AddToDateTime(&currenttime, 0, 0, 0, 0, &AgentAcceptTime, 0) >= &Current⇒
Task.EscalationTime Then
       If &TimeDelta > 0 Then
          &AgentAcceptTime = &TimeDelta;
       Else
/* The current time already exceeds the Escalationn time*/
       End-If;
    End-If;
```


Parameters

Parameter	Description
LogicalQueueId	Specify the logical queue that you want to add this task to, as a string. You can specify a null value `("")`for this parameter. If no value is provided , the logicalQueueID that is a property of task is used.
AgentID	Specify the agent ID for this task, as a string. You can specify a null value `("")` for this parameter. If you specify a null value, the task is given to any available agent. You might expect to use a null for this parameter generally, as most tasks are to be reassinged for random agent allocation. However, there may be cases where a particular agent (such as one who previously worked on this task) might be required to take this task. In this case an agentID is provided
Timeout	Specify the time out period for this task as a number of minutes. You can specify a zero for this parameter. If no value is provided, the Timeout property of the task is used. The acceptance timeout is the period of time that an agent has to accept an assigned task (click on the flashing icon on the MultiChannel console). If the task is not accepted within this time, the task is re-enqueued.
ResponseTime	Specify the response time for this task as a number of minutes. You can specify a zero for this parameter. If no value is provided, the ResponseTime property of the task is used. The response time refers to the when a task should be completed by in order to satisfy customer contracts (such as, a platinum customer who requires a response in one day for a email. If by the end of 24 hours the task is not completed, it is escalated or the response time is exceeded.)
Cost	Specify the cost of this task as a number. You can specify a zero for this parameter. This is the cost of the task as set by the application when the task was enqueued. If no cost is set at that time, then the default cost is used.
Priority	Specify the priority of the task as a number. One is the lowest possible priority. Higher numbers indicate high priorities. If no priority is set, the default priority for the task is used.
MinSkill	Specify the minimum skill level required to do this task, as a number. You can specify a zero for this parameter.

Returns

None.

Example

```
&CurrentTask = create Task(&Task_Number);
&CurrentTask.Enqueue(&CurrentTask.PhysicalQueueID, "", &AgentAcceptTime, &Task⇒
CompleteTime, &CurrentTask.Cost, &CurrentTask.Priority, &CurrentTask.SkillLevel);
```


Refresh

Syntax

`Refresh``()`

Description

Use the Refresh method to refresh the task properties. If you only want to refresh the status, use the RefreshStatus method.

Parameters

None.

Returns

None.

Example

The following refreshes all the properties of the task:

```
&MyTask.Refresh();
```

See Also

Chapter 44, "Universal Queue Classes," RefreshStatus, page 2456

RefreshStatus

Syntax

`RefreshStatus``()`

Description

Use the RefreshStatus method to refresh only the status of the task. If you want to refresh all the task properties, use the Refresh method.

Parameters

None.

Returns

None.

Example

The following refreshes the status of the task:

```
&MyTask.RefreshStatus();
```

See Also

Chapter 44, "Universal Queue Classes," Refresh, page 2456

Task Properties

In this section, we discuss the task class properties. The properties are discussed in alphabetical order.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

ApplicationData

Description

This property returns a string in which data relevent to the task is passed. Typically, for a task, it contains the URL to show when an agent access this task, the subject, and so on. This data is not really useful to the end-user, but only for administrators and for test purposes.

This property is read-only.

Comments

Description

Use this property to either set or return the comments associated with this task, as a string.

This property is read-write.

EnqueueTime

Description

This property returns the datetime when the task was submitted to the universal queue as a datetime value.

This property is read-only.

EscalationTime

Description

This property returns the datetime when the task will be removed from the queue and placed in the escalated pool.

This property is read-only.

Language

Description

This property returns the language associated with the task as a string.

This property is read-only.

OriginalTime

Description

This property returns the datetime when the task was first submitted to the universal queue as a datetime value. This value is maintained for a task through its various stages and cannot change.

This property is read-only.

OverFlowTime

Description

This property returns the datetime when the task has reached the stipulated time during which the universal queue server could not find an agent to accept the task.

This property is read-only.

PhysicalQueueID

Description

This property returns the physical queue associated with the task as a string.

This property is read-only.

TiedAgentID

Description

This property returns the agent ID of the agent this task is enqueued for as a string.

This property is read-only.

TaskList Class

Use the task list class to refresh or view a task list.

TaskList Method

This section discusses the task list method Refresh.

Refresh

Syntax

Refresh()

Description

Use the Refresh method to refresh the task list properties.

Parameters

None.

Returns

None.

Example

The following refreshes all the properties of the task list:

```
&MyTaskList.Refresh();
```

TaskList Properties

This sections discusses all the properties for the task list class. The properties are discussed in alphabetical order.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

PhysicalQueueID

Description

This property returns the physical queue associated with the task list as a string.

This property is read-only.

Task

Description

This property returns all of the tasks in the task list as an array of task objects.

This property is read-only.

Example

```
For &I = 1 to &MyTaskList.Total
   &MyTask = &MyTaskList[&I];
   /* do work on every task in list */
End-For;
```

TaskType

Description

This property returns the task type as a string. A tasklist only contains tasks of a single type. The following are the valid task types for a tasklisk:

Value	***Description***
ACPT	Accepted
ASGN	Assigned
ENQ	Enqueued
ESCL	Escalated
OVFL	Overflowed

This property is read-only.

Total

Description

This property returns the total number of tasks in the list as a number.

This property is read-only.

Util Class

The Util class is used to perform certain useful tasks, such as calculating times (so a task will timeout).

Util Methods

This section discusses the Util class methods. The methods are discussed in alphabetical order.

GetLogicalQueue

Syntax

`GetLogicalQueue`*`(PhysicalQueueID)`*

Description

Use the GetLogicalQueue to get the logical queue ID for the specified physical queue.

Parameters

Parameter	***Description***
physicalQueueID	Specify the physical queue ID that you want to find the logical queue for, as a string.

Returns

A string containing the logical queue ID.

Example

The following code example returns the logical queue ID for a given physical queue ID .

```
&LogicalQueueID = &testUtil.GetLogicalQueue(&PhysicalQueueID);
```

GetTimeDiff

Syntax

`GetTimeDiff``(`*`DateTime1`*`,`*`DateTime2`*`)`

Description

Use the GetTimeDiff method to determine the difference between two datetime values.

Parameters

Parameter	***Description***
DateTime1	Specify the starting datetime as a datetime value.
DateTime2	Specify the ending datetime as a datetime value.

Returns

A number specifying the difference between the two datetime values (in seconds.)

Example

In the following example, &nPeriod is 15 minutes.

```
&DTM1 = DateTimeValue("10/09/97 10:35:36");
&DTM2 = DateTimeValue("10/09/97 10:50:36");

Local number &nPeriod = &myUtil.GetTimeDiff(&DTM1, &DTM2) / 60;
```

MCFFactory Example

The following is an example of creating an MCFFactory, then traversing the branches to reach elements from different points. It is the loops that are important—any code within the loops is just an example of retrieval.


```
&testMCFFactory = create MCFFactory(100); /* set a maximum number of tasks in a⇒
 task  list to 100 */

Local integer &i, &j, &k;
&i = 1;
&j = 1;
&k = 1;
&l = 1;
&m = 1;
&n = 1;

While &j < &testMCFFactory.LogicalQueue.Len + 1
   /*looping through the logical queues*/
   While &i < &testMCFFactory.LogicalQueue [&j].PhysicalQueue.Len + 1
      /*looping through the physical queues*/

      /*  e.g. Tasklists by physical queues*/
      &PhysQID = &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Physical⇒
QueueID;
      &nOvflTaskTotal = &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Over⇒
FlowedTaskList.Total;

      While &k < &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].OverFlowed⇒
TaskList.Total + 1
         /*looping through the Tasklist on physical queues*/

         /*  e.g. Tasks by physical queues*/
         &tasknum = &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Over⇒
FlowedTaskList.Task [&k].TaskNumber;

         &k = &k + 1;
      End-While; /* &k loop */
      /*Task List by Agents on a Physical Queue */
      While &l < &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].TotalAgents⇒
 + 1
         While &m < &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Agent ⇒
[&l].TotalPhysicalQueues + 1
            /*  e.g. Workload of agents on physical queues.  NOTE: &m  and &i are⇒
 not the same value for the same physical queue.*/
            &AgentMaxWorkload = &testMCFFactory.LogicalQueue [&j].PhysicalQueue ⇒
[&i].Agent [&l].AgentProps [&m].Workload;
            While &n < &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Agent ⇒
[&l].AgentsTasks [&m].AcceptedTaskList.Total + 1
               /* e.g. tasks on Agent's accepted list  NOTE: &m  and &i are not⇒
 the same value for the same physical queue.*/
               &tasknum = &testMCFFactory.LogicalQueue [&j].PhysicalQueue ⇒
[&i].Agent [&l].AgentsTasks [&m].AcceptedTaskList.Task [&n];
               &n = &n + 1;
            End-While; /* &n Loop - tasks on Phys Q of Agents  */
            &n = 1;
            &m = &m + 1;
         End-While; /* &m loop - Phys Q of Agents  - convoluted*/
         &m = 1;
         &l = &l + 1;
      End-While; /* &l loop - Agents on Phys Q */
      &i = &i + 1;
      &k = 1;
   End-While; /* &i loop- Physical Queues*/
   &j = &j + 1;
   &i = 1;
End-While; /* &j loop - Logical Queues*/
```

Chapter 45

Verity Search Classes

This chapter provides an overview of Search classes and discusses the following topics:

- Search class hierarchy
- Error handling
- Search object declaration
- Scope of the search objects
- Session class method
- Search class reference section

See Also

PeopleTools 8.51 PeopleBook: System and Server Administration, "Configuring Search and Building Search Indexes"

Understanding Search Class

The Search API enables you to create a logical search index as well as access a search index and query its contents. Every search index in the PeopleSoft system must have a logical search index created to describe it.

The SearchIndex objects can be created for the following types of content:

- Indexing content references in a PeopleSoft portal
- Indexing data in PeopleSoft records
- Spidering a file system
- Spidering URL's
- User defined indexes (such as a PeopleSoft application)

Portal users can index and search on content references through the Search API using PeopleCode. The Portal Administration pages provide a GUI access to indexing a portal. This search index is what is used when doing searches on the portal. Users can create search indexes and index data in PeopleSoft records. The Search Index Designer provides a GUI tool to create these indexes.

Users can create search indexes and spider files systems or URL's. The Search Index Designer provides a GUI tool to create these indexes.

Users can create their own custom search indexes.

PeopleSoft has integrated the Verity search engine into PeopleTools.

All searches done using the Search API are executed against a Verity search index. A search index is similar to a record in many ways:

- It has fields that hold information about a document.
- It stores one row of information for each document indexed.
- It can store information in different languages.

You must build a search index before you can use the Search API.

For the portal, you can build a search index either using the Search API (BuildSearchIndex) PortalRegistry method) or using the Portal Administration.

To index and search something that isn't registered in the portal, you must build your own search index.

After you've built a search index, you can execute a search against the index, and execute a search query. A search query (SearchQuery object) represents the items in an index that match a query. These matched items are retrieved in the SearchResult collection.

In addition, you can also check for any errors that were generated by the search.

If you specify multiple indexes to search over, and none of the collections' indexes are built, an error is returned. Otherwise, if there is at least one built index collection, and there is a matching search result, it is displayed.

See Also

Chapter 45, "Verity Search Classes," BuildSearchIndex, page 2474

Using the SearchResults Collection

You specify the size of the SearchResult collection when you execute the search. If more SearchResults were returned than can be contained in the single SearchResult collection, you can access the next set of SearchResults by executing the search again, specifying only a different starting point.

For example, suppose your search returns 28 documents that match the query, but the size you specified when you executed the search was 10. You have to execute the search three times to access all the documents that matched the query.

The following pseudo-code could be behind a "Next Matches" button.

- &Start indicates at what document you want to start your search query.
- &Size is the number of search results returned in every search.

- &SearchQuery is the query object you created at the start of your program.

```
&Start = &Start + &Size;
```

```
&SearchQuery.Execute(&Start, &Size);
```

Understanding SearchResults

Each SearchResult in the SearchResult collection contains:

- Key
- Score
- SearchFields

The key uniquely identifies each document in the SearchResult collections. Keys provide a link back to the item that was indexed. The value returned by the Key property depends on the search index. If you're using a search index with the portal, the key contains the content provider and the URL. If you're not using the portal search index, the key is determined by the developer who created the search index.

Each query returns matched documents in relevance-ranked order, with those documents considered most relevant appearing at the top of the list. During search processing, a score is calculated for each retrieved document. This is the value stored in the Score property. A Score is assigned a value from 0.00 to 1.00, where 1.00 represents a perfect match to the search criteria provided.

Each SearchResult contains fields. If this is a non-portal search, these fields were specified by the developer when they created the search index. If this is a portal search, the fields are the following:

Field	***Description***
VALID_FROM_DATE	Valid from date
VALID_TO_DATE	Valid to date
CREATION_DATE	Creation date
CONTENT_PROVIDER	Content provider name
URL	URL
DESCRIPTION	Description

An external URL has the full URI attached so it might look like this:

```
URL: http://sports.mysports.com/nba/teams/chi/
```

Whereas a URL with a content provider specified is comprised of two portions.

```
Content Provider: HRMS
```



```
URL:ICType=Panel&Menu=ADMINISTER_WORKFORCE_(U.S.)&Market=GBL&PanelGroupName=⇒
ABSENCE_HISTORY1
```

You can interrogate each field in the SearchField collection to find its name and value.

See Also

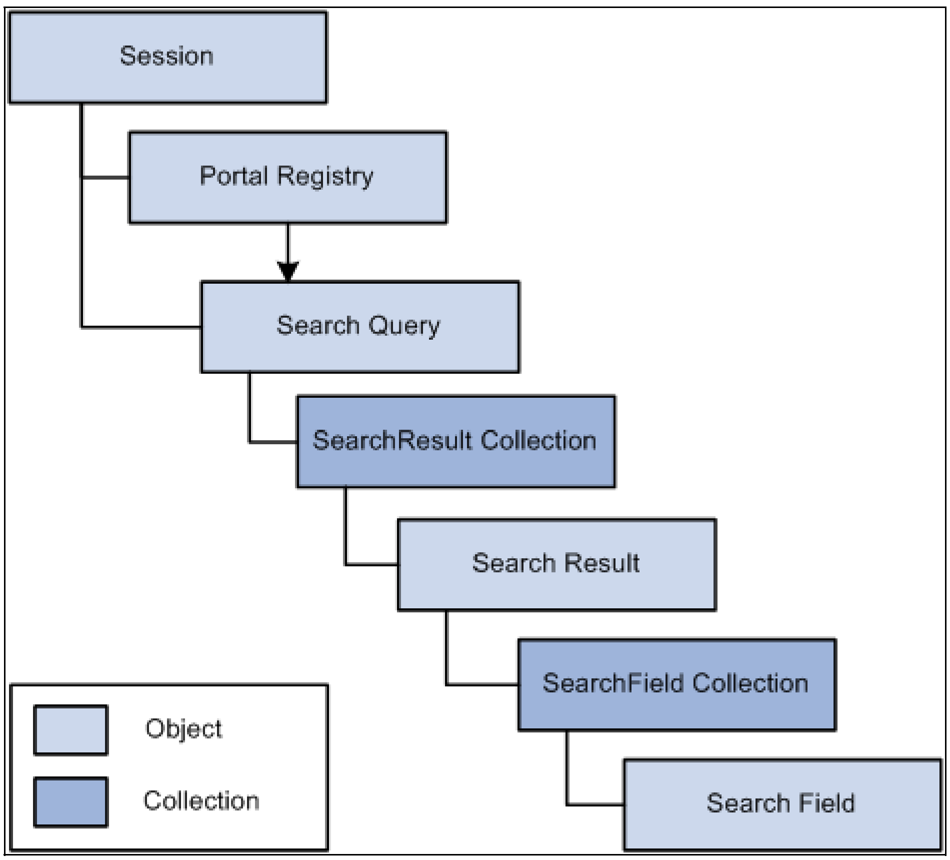
Chapter 45, "Verity Search Classes," Key, page 2487

Chapter 45, "Verity Search Classes," Score, page 2487

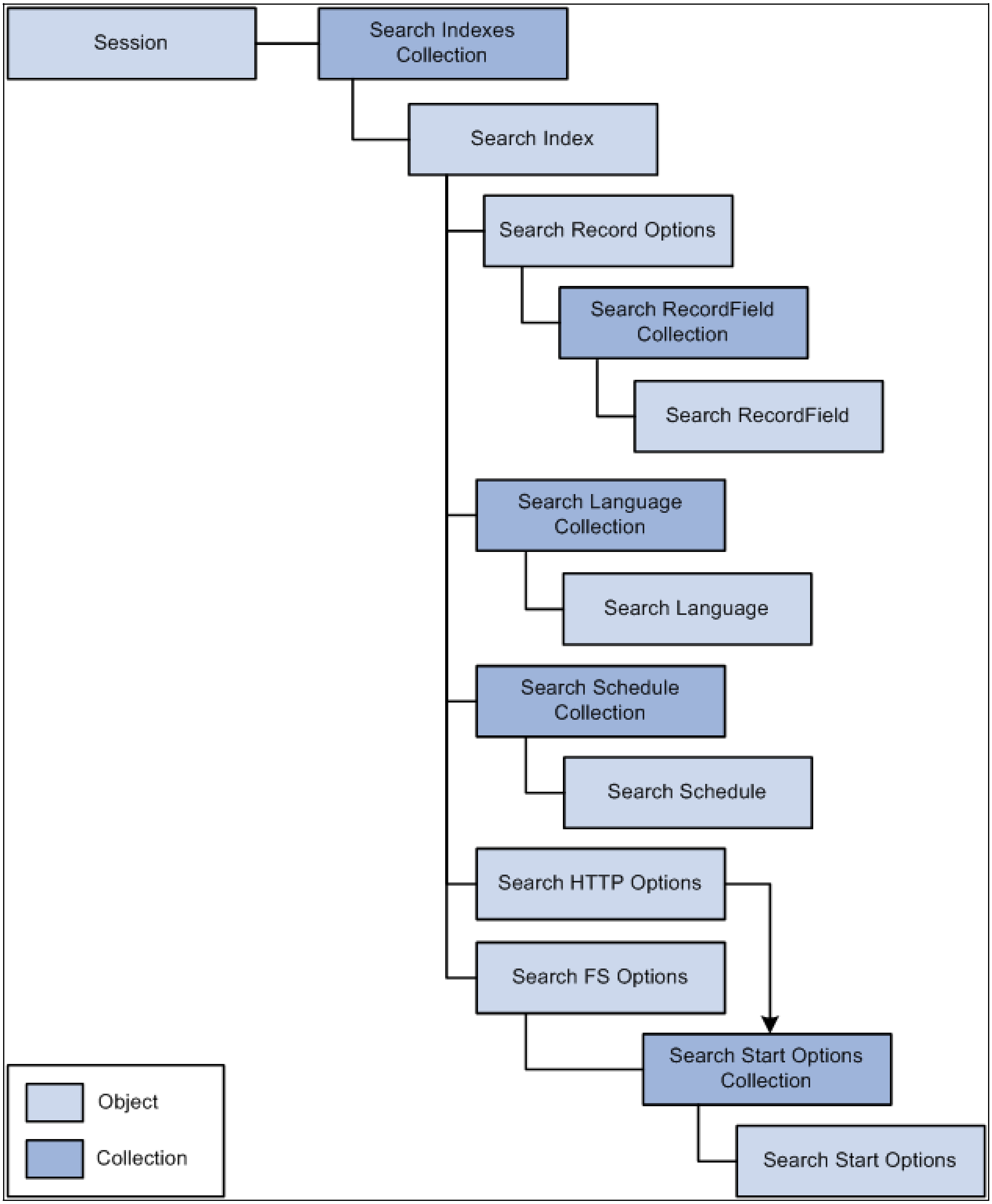
Chapter 45, "Verity Search Classes," SearchFields, page 2488

Search Classes Hierarchy

There are many different classes used with the Search API. The following flowchart shows the different classes, and how they are interrelated.



Search class hierarchy (part 1 of 2)



Search class hierarchy (part 2 of 2)

Error Handling

All errors for the Search API, like the other APIs, are logged in the PSMessages collection, instantiated from a session object.

The search classes log errors "interactively", that is, as they happen. For example, suppose you specified an invalid SearchField name. The error would be logged in the PSMessages collection as soon as you executed the ItemByName method.

If you want to search for errors in the query string before you execute the search, you can use the SearchQuery Parse method.

When to check for errors depends on your application. However, if you check for errors after every assignment, you may see a performance degradation.

The easiest way to check for errors is to check the number of messages in the PSMessages collection, using the Count property. If the Count is 0, there are no errors.

```
Local ApiObject &MySession;
Local ApiObject &ERRORCOL;
Component ApiObject &SearchQuery, &SearchResultCol, &SearchResult;
Component Number &Start, &Size;

&MySession = %Session;

If &MySession Then
   /* connection is good */

   &SearchQuery = &MySession.GetSearchQuery();
   &SearchQuery.Indexes = "PSC";
   &SearchQuery.Language = %Language;
   &SearchQuery.QueryText = SEARCH_RECORD.USER_QUESTION.Value;

   &Start = 1;
   &Size = 20;

   &SearchResultsCol = &SearchQuery.Execute(&Start, &Size);

   If &SearchResult.HitCount > 0 Then
      For &I = 1 to &SearchResultCol.Count
         &SearchResult = &SearchResultCol.Item(&I);

         /* Do processing */

      /* Do error checking */

         &ERRORCOL = &MySession.PSMessages;
         If (&ERRORCOL.Count <> 0) Then
            /* errors occurred . . . do processing */
         Else
            /* no errors */
         End-If;

      End-For;
   Else
      /* do processing for no returned matches to query */
   End-if;
Else
   /* do processing for no connection */
End-If;
```

See Also

Chapter 38, "Session Class," Error Handling, page 2180

Chapter 45, "Verity Search Classes," Parse, page 2476

Data Type of Search Objects

All search objects, like a SearchResult collection, a SearchField, and so on, are declared as type ApiObject. For example,

```
Local ApiObject &SearchResultCol;
```

```
Local ApiObject &SearchField;
```


Note. All Search objects can be declared only as Local.

Scope of the Search Objects

A search object can be instantiated only from PeopleCode using a PortalRegistry or Session object.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on. The limits placed on where you can use this object depend on how the object is used. For example, if you're doing search index administration, you are updating the database, and that can be done only in certain events.

Note. The Search API is not supported for Application Engine programs run via the PSAESRV process. In addition, the Search API does not work in Application Engine programs run on the System 390.

Session Class Methods

A SearchQuery object must be instantiated from either a session object or a Portal Registry object. The Search API doesn't have any built-in functions.

In this section, we discuss the Session class methods. The methods are discussed in alphabetical order.

See Also

Chapter 38, "Session Class," page 2179 and Chapter 30, "Portal Registry Classes," page 1587

GetSearchIndexes

Syntax

GetSearchIndexes()

Description

The GetSearchIndexes method returns a collection of SearchIndexes objects.

A SearchIndexes object and its children are used only for index administration, that is, creating and building search indexes. This method is *not* used with queries.

Parameters

None.

Returns

A SearchIndexes Collection object, populated with zero or more SearchIndex objects.

See Also

Chapter 45, "Verity Search Classes," SearchIndexes Collection, page 2491

GetSearchQuery

Syntax

`GetSearchQuery``()`

Description

The GetSearchQuery method returns an empty search query object used to start a search.

Parameters

None.

Returns

An empty search query object.

Example

```
&SearchQuery = %Session.GetSearchQuery();
```

See Also

Chapter 45, "Verity Search Classes," GetSearchQuery, page 2474

PortalRegistry Class Methods

A SearchQuery object must be instantiated from either a session object or a Portal Registry object. The Search API doesn't have any built-in functions.

In this section, we discuss the PortalRegistry class methods. The methods are discussed in alphabetical order.

See Also

Chapter 32, "Process Request Classes," page 1835

BuildSearchIndex

Syntax

`BuildSearchIndex`*`(Language)`*

Description

The BuildSearchIndex method builds a portal search index for the specified language.

Search Collection Path Considerations

The Search Collection Path field in the configuration file points to a directory where collections can be built and queried through the search API. However, the BuildSearchIndex method *always* builds the collection in the default path, that is, in the following:

PS_HOME\data\search*portal_name**db_name**lang_cd*

Parameters

Parameter	*Description*
Language	Specify the language code that you want the search index built in.

Returns

An optional Boolean value: True if the search index is built successfully, False otherwise.

GetSearchQuery

Syntax

`GetSearchQuery``()`

Description

The GetSearchQuery method returns an empty search query object.

Note. When you use this method to get a search query, the index property is already set to the portal registry search index, and the language property is defaulted to the end-user's language.

Parameters

None.

Returns

An empty SearchQuery object with the index and language property already set.

Example

```
&Portal = %Session.GetPortalRegistry();

&Portal.Open("PORTAL");

&SearchQuery = &Portal.GetSearchQuery();
```

See Also

Chapter 45, "Verity Search Classes," GetSearchQuery, page 2473

SearchQuery Class

The SearchQuery object is used to specify the search index and execute a query against it.

A SearchQuery object is returned by the following:

- The GetSearchQuery method from a PortalRegistry object.
- The GetSearchQuery method from a Session object.

See Chapter 45, "Verity Search Classes," GetSearchQuery, page 2474.

See Chapter 45, "Verity Search Classes," GetSearchQuery, page 2473.

SearchQuery Class Methods

In this section, we discuss the SearchQuery class methods.

Execute

Syntax

`Execute``(Start,Size)`

Description

The Execute method runs a search query beginning at the document number specified by *Start*, and returns the number of results (or less) specified by *Size* in a SearchResult Collection.

Parameters

Parameter	***Description***
Start	Specify the document with which you want to start your query search. This parameter takes an integer, 1 or higher.
Size	Specify the number of results that you want returned. This parameter takes an integer value.

Returns

A SearchResult collection.

Example

```
&SearchResultCol = &SearchQuery.Execute(1, 20);
```

See Also

Chapter 45, "Verity Search Classes," SearchResult Collection, page 2485

Parse

Syntax

`Parse``()`

Description

Use the Parse method to verify the syntax of the query string specified through the QueryText property for syntax errors, such as whether there is a missing quotation mark or closing bracket.

The Parse method does not produce a collection of search results. If there are any errors, it returns a reference to a ParseResult collection that you can search through for errors. If there are not any errors, it returns a Null object.

Parameters

None.

Returns

A reference to a ParseResult collection if there are errors, Null otherwise.

Example

```
Local ApiObject &MySession;
Local ApiObject &ParseResults;
Component ApiObject &SearchQuery, &SearchResultCol, &SearchResult;
Component Number &Start, &Size;

&MySession = %Session;

If &MySession Then
   /* connection is good */

   &SearchQuery = &MySession.GetSearchQuery();
   &SearchQuery.Indexes = "PSC";
   &SearchQuery.Language = %Language;
   &SearchQuery.QueryText = SEARCH_RECORD.USER_QUESTION.Value;

   &ParseResults = &SearchQuery.Parse();

   If All(&ParseResults) AND &ParseResults.ErrorCount > 0 OR &Parse⇒
Results.WarningCount > 0 Then

      /* Process results */

   End-if;
End-if;
```

See Also

Chapter 45, "Verity Search Classes," ParseResult Collection, page 2482

SearchQuery Class Properties

In this section, we discuss the SearchQuery class properties. The properties are discussed in alphabetical order.

HitCount

Description

The HitCount property returns the total number of documents that actually match the query.

This property is read-only.

Indexes

Description

Use the Indexes property to specify the list of indexes you want the search to query. This list is in the form of a list of comma separated index names. Use the Execute method to execute the query on all the indexes specified in this property.

If you want to search against a single index, specify the index name without any comma.

This property is read-write.

Example

The following example uses two search indexes called CATALOG1 and CATALOG2 for the word "bicycle", with the requested fields called "DESCRIPTION" and "PRICE" in the results.


```
&sqQuery = %Session.GetSearchQuery();
&sqQuery.Indexes = "CATALOG1, CATALOG2";
&sqQuery.Language = %Language;
&sqQuery.Querytext = "bicycle";
&sqQuery.SortSpecifications = "PRICE desc DESCRIPTION asc";
&sqQuery.RequestedFields = "PRICE, DESCRIPTION";

/* Execute the search, getting the first 20 results. */
&srcResults = &sqQuery.Execute(1, 20);
If &srcResults <> Null And &srcResults.Count > 0 Then

&srCurrent = &srcResults.First();
Repeat
       Local ApiObject &srFieldColl;
&srFieldsColl = &srCurrent.SearchFields;
       Local ApiObject &srField = &srFieldColl.First();
 Repeat
      /* Do something with this field. */
         &srField = &srFieldColl.Next();
       Until None(&srField);
   Until None(&srCurrent);
Else
   ReportSearchAPIErrors();
   Local string &msg;
   &msg = MsgGetText(145, 40, "No results for your search");
End-If;

Function ReportSearchAPIErrors()
   Local ApiObject &msgColl = %Session.PSMessages;
   Local number &i;
   For &i = 1 To &msgColl.count
      Local ApiObject &msg = &msgColl.item(&i);
      Error (&msg.Text);
   End-For;
   &msgColl.DeleteAll();
End-Function;
```

IndexName

Description

Note. This property remains for backward compatibility only. Use the Indexes property instead.

See Chapter 45, "Verity Search Classes," Indexes, page 2478.

The IndexName property specifies the SearchIndex this query is to retrieve the result from.

You *must* set this property before a query can be performed on the SearchQuery object returned from the GetSearchQuery method.

This property is read-write.

Example

```
&SearchQuery.IndexName = "PSA";
```


KnowledgeBase

Description

Use the KnowledgeBase property to specify a topicset (that is, a file generated by using the mktopics Verity command for a search query. This property takes the name of a file, as a string.

This property is read-write.

Language

Description

The Language property specifies the language of the search index.

The default value for this property is the current user's language.

This property is read-write.

ProcessedCount

Description

The ProcessedCount property returns the total number of documents that were subjected to the search query.

This property is read-only.

QueryText

Description

The QueryText property returns the query text to be submitted to the SearchQuery object.

This property is read-write.

RequestedFields

Description

The RequestedFields property specifies the set of fields to be returned with the results.

The value of this property is a list of fields, separated by commas.

If this property is not specified, all the fields (except a few internal fields) are returned with the result. You may get better performance by specifying only the fields you want returned.

This property is read-write.

See Also

Chapter 45, "Verity Search Classes," SortSpecifications, page 2481 and Chapter 45, "Verity Search Classes," SearchField Collection, page 2488

ScorePrecision

Description

The ScorePrecision property specifies how precise the score should be displayed with the results.

The value of this property is a string, and can be one of the following (case is insensitive)

- 8bit
- 16bit
- 32bit

The system displays two digits after decimal point if "8bit" is selected, and 4 digits if "16bit" is selected. The default value of this property is "8bit".

If this property is set to any other value, the Execute method generates an error and terminates the search.

This property is read-write.

SortSpecifications

Description

The SortSpecifications property specifies how the results of a search should be sorted.

This property should be in the following format:

`fieldname1 {asc | desc} [fieldname2 {asc | desc} ...]`

where *fieldname1* and *fieldname2* are field names to be used to sort the results and **asc** and **desc** are keywords for sorting in ascending or descending order.

Specifying multiple fields allows you to perform a secondary sort within the primary sorted results. For example, specifying `price asc descr desc` sorts the returned results by price in ascending order. Within the price, the results are sorted by descr field in descending order.

The default value of this property is to sort the results by score field in descending order.

Note. If you specify this property, you must also specify the RequestedFields property.

See Also

Chapter 45, "Verity Search Classes," RequestedFields, page 2480

ParseResult Collection

A ParseResult Collection is returned from the Parse SearchQuery method.

See Chapter 45, "Verity Search Classes," Parse, page 2476.

ParseResult Collection Methods

In this section, we discuss the ParseResult collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first ParseResult object in the ParseResult collection. If the ParseResult collection is empty, it returns Null.

Example

```
&MyResult = &MyParseResultCollection.First();
```

Next

Syntax

Next()

Description

The Next method returns the next ParseResult object in the ParseResult collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

This method returns a Null when there are no more ParseResult objects in the collection.

Parameters

None.

Returns

Either a reference to a ParseResult object, or, if there are no more results in the ParseResult collection, a Null.

Example

```
Repeat
   /* &parseInfo.Message() contains the message, process it */
   &parseInfo.Next()
Until None(&parseInfo);
```

ParseResult Collection Properties

In this section, we discuss the ParseResult collection properties The properties are discussed in alphabetical order.

ErrorCount

Description

The ErrorCount property returns the number of ParseResult objects in this collection with a severity of ERROR.

This property is read-only.

Example

```
For &I = 1 to &ParseResults.ErrorCount
   /* process message */
End-For;
```

WarningCount

Description

The WarningCount property returns the number of ParseResult objects in this collection with a severity of WARNING.

This property is read-only.

ParseResult Class

A ParseResult object is returned from either the First or Next ParseResult Collection methods.

See Chapter 45, "Verity Search Classes," ParseResult Collection, page 2482.

ParseResult Class Properties

In this section, we discuss the ParseResult class properties The properties are discussed in alphabetical order.

Message

Description

The Message property returns the text of the error or warning message as a string.

This property is read-only.

Severity

Description

The Severity property returns the severity of the message as a string. Values are:

Value	***Description***
ERROR	Message is an error message.
WARNING	Message is a warning message.

This property is read-only.

SearchResult Collection

A SearchResult collection is returned from the Execute SearchQuery method.

See Chapter 45, "Verity Search Classes," Execute, page 2476.

SearchResult Collection Methods

In this section, we discuss the SearchResult collection methods. The methods are discussed in alphabetical order.

First

Syntax

`First``()`

Description

The First method returns the first SearchResult object in the SearchResult collection. If the SearchResult collection is empty, it returns Null.

Example

```
&MyResult = &MyCollection.First();
```

Item

Syntax

`Item`*`(number)`*

Description

The Item method returns the SearchResult object with the position in the SearchResult collection specified by *number*.

Parameters

Parameter	*Description*
Number	Specify the position number in the collection of the SearchResult object that you want returned. Values start at 1.

Returns

A SearchResult object if successful, Null otherwise.

Example

```
For &I = 1 to &MyCollection.Count

   &MyResult = &MyCollection.Item(&I);

   /* Do processing */

End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next SearchResult object in the SearchResult collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MyResult = &MyCollection.Next();
```

SearchResult Collection Property

In this section, we discuss the SearchResult collection property Count.

Count

Description

This property returns the number of SearchResult objects in the SearchResult collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SearchResult Class

The SearchResult object represents a specific SearchResult in a SearchResult Collection object.

SearchResult objects are instantiated from a SearchResult Collection with the First, ItemByName or Next methods.

See Chapter 45, "Verity Search Classes," SearchResult Collection, page 2485.

SearchResult Class Properties

In this section, we discuss the SearchResult class properties. The properties are discussed in alphabetical order.

Key

Description

The Key property returns the key for this SearchResult object.

This property is read-only.

Score

Description

The Score property returns the score for this SearchResult object as a string. The decimal separator in this string depends on the Peoplesoft language that was used for search. For example, if you searched in Spanish, the decimal separator would be a ','.

Use this property to display the score.

If you convert this string to a number using the Value function, you might not receive the correct result. This is because the Value function always expects the decimal separator character to be a period. You need to explicitly replace the decimal separator with a period before using the Value function. To avoid such conversion problems, if you want to perform score arithmetic, such as evaluating a score to see if it's below a threshold, use the ScoreAsNumber property.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," ScoreAsNumber, page 2488

ScoreAsNumber

Description

This property returns the relevancy score as a floating point number between 0 and 1. This property is the same as the Score property, except that this property returns the score as a floating point number instead of a string. Use this value to perform score arithmetic such as comparing it to a threshold value. If you want to display the score, use the Score property.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," Score, page 2487

SearchFields

Description

The SearchFields property returns a SearchField Collection.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," SearchField Collection, page 2488

SearchField Collection

A SearchField collection is returned by the SearchFields SearchResult property.

See Chapter 45, "Verity Search Classes," SearchFields, page 2488.

SearchField Collection Methods

In this section, we discuss the SearchField collection methods. The methods are discussed in alphabetical order.

First

Syntax

`First``()`

Description

The First method returns the first SearchField object in the SearchField collection. If the SearchField collection is empty, it returns Null.

Example

```
&MyField = &MyCollection.First();
```

ItemByName

Syntax

`ItemByName``(`*`Name`*`)`

Description

The ItemByName method returns the SearchField object specified by *Name*.

Parameters

Parameter	***Description***
Name	Specify the name of the SearchField you want returned.

Returns

A SearchField object if successful, Null otherwise.

Example

```
&MyField = &MyCollection.ItemByName(VALID_TO_DATE);
```

Next

Syntax

Next()

Description

The Next method returns the next SearchField object in the SearchField collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MyField = &MyCollection.Next();
```

SearchField Collection Property

In this section, we discuss the SearchField collection property Count.

Count

Description

This property returns the number of SearchField objects in the SearchField collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SearchField Class

A SearchField is returned by First, ItemByName, and Next SearchField Collection methods.

See Chapter 45, "Verity Search Classes," SearchField Collection, page 2488.

SearchField Class Properties

In this section, we discuss the SearchField class properties. The properties are discussed in alphabetical order.

Name

Description

The Name property returns the name of this SearchField object as a string.

This property is read-only.

Value

Description

The Value property returns the value for this SearchField object as a string.

This property is read-only.

SearchIndexes Collection

A SearchIndexes collection is returned by the GetSearchIndexes Session class property.

The SearchIndexes in this collection represent all of the *logical* search indexes that have been created in the PeopleSoft system.

Every search index built and used in a PeopleSoft system must have a logical search index associated with it.

See Chapter 45, "Verity Search Classes," GetSearchIndexes, page 2472.

SearchIndexes Collection Methods

In this section, we discuss the SearchIndexes collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first SearchIndex object in the SearchIndexes collection. If the SearchIndex collection is empty, it returns Null.

Example

```
&MySearchI = &MyCollection.First();
```

ItemByName

Syntax

ItemByName*(Name)*

Description

The ItemByName method returns the SearchIndex object specified by *Name*.

Parameters

Parameter	***Description***
Name	Specify the name of the SearchIndex that you want returned.

Returns

A SearchIndex object if successful, Null otherwise.

Example

```
&MySearchI = &MyCollection.ItemByName("MySearchIndex");
```

Next

Syntax

Next()

Description

The Next method returns the next SearchIndex object in the SearchIndexes collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MySearchI = &MyCollection.Next();
```

SearchIndex Collection Property

In this section, we discuss the SearchIndex collection property Count.

Count

Description

This property returns the number of SearchIndex objects in the SearchIndexes collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SearchIndex Class

A SearchIndex is returned by First, ItemByName, and Next SearchIndexes Collection methods.

A SearchIndex represents a logical search index, and all the information describing it, in the PeopleSoft system. Every search index built and used in a PeopleSoft system must have a logical search index associated with it.

See Chapter 45, "Verity Search Classes," SearchIndexes Collection, page 2491.

SearchIndex Class Method

In this section, we discuss the SearchIndex class method Save.

Save

Syntax

```
Save()
```

Description

Use the Save method to save any changes to the search index.

Note. If you change a value of one of the properties of the subobjects (such as some of the search options, and so on) the change won't be written to the database until the SearchIndex object has been saved.

Parameters

None.

Returns

A Boolean value: True if the SearchIndex object is saved successfully, False otherwise.

SearchIndex Class Properties

In this section, we discuss the SearchIndex class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

ExtraOptions

Description

This property is used for the spidering search indexes. Parameters that are not covered in the SearchIndex object and its children can be passed to the Verity spidering utility using this property.

This property is read-write.

FSOpts

Description

This property returns a reference to a Search FS Options object, for FSYS type search indexes.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," Search FS Options Class, page 2518

HTTPOpts

Description

This property returns a reference to a Search HTTP Options object, for HTTP type search indexes.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," Search HTTP Options Class, page 2514

Languages

Description

This property returns a reference to the Search Language collection.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," Search Language Collection, page 2505

Location

Description

This property specifies the file system location where the search index (collection) is located. You must use a full path name.

This property is read-write.

Name

Description

This property returns the name of the SearchIndex, as a string.

This property is read-only.

RecOpts

Description

This property returns a reference to a SearchRecord Options collection, for RECD type search indexes.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," SearchRecord Options Class, page 2497

Schedules

Description

This property returns a reference to the Search Schedule collection.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," Search Schedule Collection, page 2509

Type

Description

The Type property specifies the type of index. This property takes a string value. The values are:

Value	*Description*
RECD	An index generated from a PeopleSoft record
HTTP	An index generated from a spider search of the World Wide Web (WWW)
FSYS	An index generated from a spider search of a file system
PRTL	An index generated from a portal
USER	A user defined index (that is, not one of the previous indexes)

This property is read-write.

SearchRecord Options Class

A reference to a search record options object is returned by the RecOpts SearchIndex property.

This object is valid only for Search Index types of RECD. These types of indexes are used to index data in a PeopleSoft database. The properties used with this object represent the values that must be set for this type of search index.

See Chapter 45, "Verity Search Classes," RecOpts, page 2496.

SearchRecord Options Class Properties

In this section, we discuss the SearchRecord option class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

Fields

Description

This property returns a reference to a SearchRecordField collection.

The SearchRecordField collection describes the records and fields that are indexed.

The maximum number of fields returned as part of this collection is 50.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," SearchRecordField Collection, page 2500

Filter

Description

This property specifies a filter (that is, a SQL WHERE clause) used to filter out unwanted records from being indexed, as a string.

This property is read-write.

IncrementalView

Description

This property specifies a view used for accessing the records that are to be used for incremental indexing, as a string.

This property is read-write.

RecName

Description

This property returns the record name for the primary record that the search index is based on. Multiple records can be used, but they must all share a common key structure. The record name returned with this property specifies this common base key structure.

This property is read-only.

VeggieKey

Description

This property specifies a replacement pattern for the VdkVgwKey (VeggieKey) used by Verity to uniquely identify a document. The setting of this property determines what is filled in for the unique identifier for each document that gets indexed. Text in this string is replaced with either literal text in the unique identifier or a value determined by one of the following tags:

Tag	***Description***
<pairs/>	Inserts a string of NAME=VALUE; pairs for each key on the Record being indexed
<row/>	Inserts the Record keys in a SQL-like syntax
<field fieldname='*MYFIELD*'/>	Inserts the value of *MYFIELD* for the Row being indexed (if it exists in the record; an error occurs when you save the data if it is missing)
<sql stmt='*SQL STATEMENT*'/>	Inserts the value returned by the SQL statement. Only the first row returned is accepted. SQL statements that return more than one column cause an error at runtime.

To use these tags, you include them as part of the VeggieKey string. The indexing process replaces them with values. For example, given the following Row of data:

```
Record: PROD_SRCH
Field: PRODUCTID=A11111 (a key on the record)
Field: DESCR="Baby goldfish"
Field: PRICE=5.00
```

You could produce any of the following unique identifiers with various settings of VeggieKey:

VeggieKey Setting	***String Produced***
"<pairs/>"	"PRODUCTID=A11111;"
"SELECT * FROM <row/>"	"SELECT * FROM PS_PROD_SRCH WHERE PRODUCTID='A11111'"
"<pairs/> <field filedname='DESCR'/>"	"PRODUCTID=A11111; Baby goldfish"
"Total price: <sql stmt='SELECT SUM(PRICE) FROM PS_PROD_SRCH'/>"	"Total price: 550.78"

Note that the last example is not truly unique, although it is legal. It is up to the application to make sure that the setting of VeggieKey produces a unique string for each indexed document, as the indexing process does not perform this check. Applications that use VdkVgwKey to identify the results of search queries cannot find the correct document if the documents do not have a unique key.

This property is read-only.

ZoneOptions

Description

This property specifies the zone option for the index as a string. Values are:

Value	***Description***
ONE	All the fields for each row of data are inserted into the same zone in the word index. This is a simple scheme which does not allow you to search only parts of the word index but may be appropriate for applications without easily-differentiated text regions.
FLD	One zone is created for each field. The data in that field for each row is inserted into the same zone. This enables applications to search only a specific part of the word index at a time, with a potential performance gain. This does not happen automatically, however; the query interface built by the application must support searching in specific zones. The name of the zone created is the name of the field. The build process may take longer with this option set.

This property is read-only.

SearchRecordField Collection

A SearchRecordField collection is returned from the Fields SearchRecord Options property.

The SearchRecordField collection describes the records and fields that are indexed.

The maximum number of fields returned as part of this collection is 50.

See Chapter 45, "Verity Search Classes," Fields, page 2498.

SearchRecordField Collection Methods

In this section, we discuss the SearchRecordField collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem*(Record, Field)*

Description

Use the DeleteItem method to delete the SearchRecordField specified by the record and field name.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

Parameters

Parameter	***Description***
Record	Specify a record name as a string.
Field	Specify a field name as a string.

Returns

A Boolean value: True, item was deleted successfully, False otherwise.

First

Syntax

First()

Description

The First method returns the first SearchRecordField object in the SearchRecordField collection. If the SearchRecordField collection is empty, it returns Null.

Example

```
&MySearchField = &MyCollection.First();
```

InsertItem

Syntax

`InsertItem``(Record, Field)`

Description

Use the InsertItem method to insert a SearchRecordField object into the SearchRecordField collection.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

If the item you are trying to insert already exists, this method returns Null.

Parameters

Parameter	***Description***
Record	Specify a record name as a string.
Field	Specify a field name as a string.

Returns

A reference to a new SearchRecordField object if successful, Null otherwise.

ItemByName

Syntax

`ItemByName``(Record, Field)`

Description

The ItemByName method returns the SearchRecordField object specified by the record and field name.

Parameters

Parameter	***Description***
Record	Specify a record name as a string.

Parameter	*Description*
Field	Specify a field name as a string.

Returns

A SearchRecordField object if successful, Null otherwise.

Example

```
&MySearchField = &MyCollection.ItemByName("SEARCH", "DATE");
```

Next

Syntax

Next()

Description

The Next method returns the next SearchRecordField object in the SearchRecordField collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MySearchOpts = &MyCollection.Next();
```

SearchRecordField Collection Property

In this section, we discuss the SearchRecordField collection property Count.

Count

Description

This property returns the number of SearchRecordField objects in the SearchRecordField collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SearchRecordField Class

A SearchRecordField is returned by the First, ItemByName, InsertItem, and Next SearchRecordField collection methods.

A SearchRecordField object describes the records and fields that are indexed.

See Chapter 45, "Verity Search Classes," SearchRecordField Collection, page 2500.

SearchRecordField Class Properties

In this section, we discuss the SearchRecordField class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

FieldName

Description

This property returns the field name for this SearchRecordField as a string.

This property is read-only.

IsAttachment

Description

This property specifies whether the search record field is an attachment. This property takes a Boolean value: True, the field is an attachment, False otherwise.

This property is read-write.

IsVerityField

Description

This property specifies whether the search record field is a Verity field. This property takes a Boolean value: True, the field is a Verity field, False otherwise.

This property is read-write.

IsWordIndex

Description

This property specifies whether the search record field is a word index. This property takes a Boolean value: True, the field is a word index, False otherwise.

This property is read-write.

RecordName

Description

This property returns the name of the record as a string.

This property is read-only.

Search Language Collection

A reference to a Search Language collection is returned by the Languages property.

See Chapter 45, "Verity Search Classes," Languages, page 2495.

Search Language Collection Methods

In this section, we discuss the Search Language collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

`DeleteItem``(`*`Name`*`)`

Description

Use the DeleteItem method to delete the Search Language specified by *Name.*

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

Parameters

Parameter	***Description***
Name	Specify the name of the Search Language that you want to delete.

Returns

A Boolean value: True, item was deleted successfully, False otherwise.

First

Syntax

`First``()`

Description

The First method returns the first Search Language object in the Search Language collection. If the Search Language collection is empty, returns Null.

Example

```
&MyLang = &MyCollection.First();
```

InsertItem

Syntax

`InsertItem`*`(LanguageCode, MapLanguageCode)`*

Description

Use the InsertItem method to insert a Search Language object into the Search Language collection.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

If the item you are trying to insert already exists, this method returns Null.

Parameters

Parameter	***Description***
LanguageCode	Specify a language code as a string.
MapLanguageCode	Specify a language code to map *LanguageCode* to (typically this is the same as *LanguageCode).*

Returns

A reference to a new Search Language object if successful, Null if not successful.

ItemByName

Syntax

`ItemByName`*`(LanguageCode)`*

Description

The ItemByName method returns the Search Language object specified by *LanguageCode.*

Parameters

Parameter	*Description*
LanguageCode	Specify the language code of the object that you want to retrieve.

Returns

A reference to a Search Language object if successful, Null otherwise.

Example

```
&MyLang = &MyCollection.ItemByName("ENG");
```

Next

Syntax

Next()

Description

The Next method returns the next Search Language object in the Search Language collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MyLang = &MyCollection.Next();
```

Search Language Collection Property

In this section, we discuss the Search Language collection property Count.

Count

Description

This property returns the number of Search Language objects in the Search Language collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Search Language Class

A reference to a Search Language object is returned by the First, ItemByName, InsertItem, and Next Search Language Collection methods.

See Chapter 45, "Verity Search Classes," Search Language Collection, page 2505.

Search Language Class Properties

In this section, we discuss the Search Language class properties. The properties are discussed in alphabetical order.

LanguageCd

Description

This property specifies the language code for the Search Language object as a string.

This property is read-only.

MapLanguageCd

Description

Specify a language code to map the given language to (typically this is the same as LanguageCode).

This property is read-only.

Search Schedule Collection

A reference to a Search Schedule collection is returned by the Schedules SearchIndex property.

See Chapter 45, "Verity Search Classes," Schedules, page 2496.

Search Schedule Collection Methods

In this section, we discuss the Search Schedule collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem*(Name)*

Description

Use the DeleteItem method to delete the Search Schedule specified by *Name.*

This method is not executed automatically. It is executed only when the SearchIndex is saved.

Parameters

Parameter	***Description***
Name	Specify the name of the Search Schedule that you want to delete.

Returns

A Boolean value: True, item was deleted successfully, False otherwise.

First

Syntax

First()

Description

The First method returns the first Search Schedule object in the Search Schedule collection. If the Search Schedule collection is empty, this method returns Null.

Example

```
&MySch = &MyCollection.First();
```

InsertItem

Syntax

InsertItem*(RunCntrlID)*

Description

Use the InsertItem method to insert a Search Schedule object into the Search Schedule collection.

This method is not executed automatically. It is executed only when the SearchIndex is saved.

If the item you are trying to insert already exists, this method returns Null.

Parameters

Parameter	*Description*
RunCntrlId	Specify the run control ID of the search schedule object as a string.

Returns

A reference to a new Search Schedule object if successful, Null if not successful.

ItemByName

Syntax

ItemByName*(RunCntrlId)*

Description

The ItemByName method returns the Search Schedule object specified by *RunCntrlId.*

Parameters

Parameter	*Description*
RunCntrlId	Specify the run control ID of the search schedule object as a string.

Returns

A Search Schedule object if successful, Null otherwise.

Example

```
&MySchedule = &MyCollection.ItemByName("MySched");
```

Next

Syntax

Next()

Description

The Next method returns the next Search Schedule object in the Search Schedule collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MySched = &MyCollection.Next();
```

Search Schedule Collection Property

In this section, we discuss the Search Schedule collection property Count.

Count

Description

This property returns the number of Search Schedule objects in the Search Schedule collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Search Schedule Class

A reference to a Search Schedule object is returned by the First, ItemByName, InsertItem, and Next Search Schedule Collection methods.

See Chapter 45, "Verity Search Classes," Search Schedule Collection, page 2509.

Search Schedule Class Properties

In this section, we discuss the Search Schedule class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

BuildType

Description

This property specifies how you want the index rebuilt with this schedule item. Values are:

Value	***Description***
INCR	Incremental updates an existing index. Use this schedule item for indexes with small amounts of data, and for cleaning up large indexes periodically when they start to fragment.
RBLD	Rebuild deletes the index and starts from scratch. Use this to add to an existing, large index.

This property is read-write.

RunCntlId

Description

This property returns the Run Control ID associated with the schedule item.

This property is read-only.

RunRecurrence

Description

This property specifies the name of the Recurrence Definition as a string. For example: "Daily Search Rebuild".

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Defining Jobs and JobSets," Creating Job Definitions

ServerName

Description

This is the name of the Process Scheduler Server that processes the requests for this schedule item, for example, "PSNT".

This property is read-write.

See Also

Enterprise PeopleTools 8.51 PeopleBook: PeopleSoft Process Scheduler, "Setting Server Definitions"

Search HTTP Options Class

A reference to a Search HTTP Options object is returned by the HTTPOpts SearchIndex property.

This object is valid only for Search Index types of HTTP. These types of indexes spider URLs. The properties available using this object represent values that must be set for spidering URL's.

See Chapter 45, "Verity Search Classes," HTTPOpts, page 2495.

Search HTTP Options Class Properties

In this section, we discuss the Search HTTP Option class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

AllowHTTPS

Description

This property is used to cause the HTTP Spider Gateway to descend HTTPS (SSL) links and HTTP links when it builds the index. This property takes a Boolean value: True, use the HTTPS (SSL) links, False otherwise.

This property is read-write.

DomainLimit

Description

This property limits spidering (indexing) to the specified domain. This property takes a string value.

This property is read-write.

GlobList

Description

This property specifies a list of filename "globs" which the indexing process uses to filter its list. The GlobListType determines how the filtering is done. The value of this property is a space-separated list of filenames with or without wildcards. The following is an example of the value for this property:

```
"*.doc *.txt *.html robots.txt"
```

This property is read-write.

See Also

Chapter 45, "Verity Search Classes," GlobListType, page 2515

GlobListType

Description

This property determines how the values specified with GlobList are used. Values are:

Value	*Description*
ALL	Ignore the values specified with GlobList. Index all filenames encountered regardless of their pattern.
EXC	Exclude the values specified with GlobList. Index everything except names that match a pattern in GlobList.
INC	Include the values specified with GlobList. Index only filenames that match a pattern in GlobList and exclude all other filenames.

This property is read-write.

See Also

Chapter 45, "Verity Search Classes," GlobList, page 2515

LinkDepth

Description

This property determines how many links the HTTP Spider descends from its starting point as a number.

A value of 1, for example, indexes the original page as well as the documents linked to by that page, and then stop.

A value of 2 indexes the original page, and documents linked to it, as well as documents linked to by the second page.

PeopleSoft recommends not setting this value over 10 without combining it with the DomainLimit property as the amount of data retrieved into the index increases geometrically with the LinkDepth.

This property is read-write.

See Also

Chapter 45, "Verity Search Classes," DomainLimit, page 2515

MIMEList

Description

This property is used in a similar fashion to GlobList. It is a space-separated list of MIME Types. The MIMEListType determines how they affect indexing. The following is an example of a value for this property:

```
"text/html text/plain application/*"
```


This property is read-write.

See Also

Chapter 45, "Verity Search Classes," MIMEListType, page 2517 and Chapter 45, "Verity Search Classes," GlobList, page 2515

MIMEListType

Description

This property is used like GlobListType, but applies to MIMEList instead of GlobList. Instead of testing the pattern of the filename, this property tests the MIME-type detected for the document when determining whether to index it.

This property is read-write.

See Also

Chapter 45, "Verity Search Classes," MIMEList, page 2516 and Chapter 45, "Verity Search Classes," GlobListType, page 2515

ProxyHost

Description

This property specifies the hostname or IP address of the HTTP proxy, if the HTTP Spider Gateway needs an HTTP proxy to connect to remote websites during the indexing process.

This property is read-write.

ProxyPort

Description

This property specifies the port number of the HTTP proxy set in ProxyHost as a number.

This property is read-write.

StartOpts

Description

This property returns a reference to a Search Start Options collection.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," Search Start Options Collection, page 2520

Search FS Options Class

A reference to a Search FS Options object is returned by the FSOpts SearchIndex property.

This object is valid only for Search Index types of FSYS. These types of indexes spider a file system. The properties available using this object represent the values that must be set for spidering a file system.

See Chapter 45, "Verity Search Classes," FSOpts, page 2495.

Search FS Options Class Properties

In this section, we discuss the Search FS Option class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

GlobList

Description

This property specifies a list of filename "globs" which the indexing process uses to filter its list. The GlobListType determines how they affect indexing. The value of this property is a space-separated list of filenames with or without wildcards. The following is an example of the value for this property:

```
"*.doc *.txt *.html robots.txt"
```

This property is read-write.

See Also

Chapter 45, "Verity Search Classes," GlobListType, page 2519

GlobListType

Description

This property determines how the values specified with GlobList are used. Values are:

Value	***Description***
ALL	Ignore the values specified with GlobList. Index all filenames encountered regardless of their pattern.
EXC	Exclude the values specified with GlobList. Index everything except names that match a pattern in GlobList.
INC	Include the values specified with GlobList. Index only filenames that match a pattern in GlobList and exclude all other filenames.

This property is read-write.

See Also

Chapter 45, "Verity Search Classes," GlobList, page 2518

MIMEList

Description

This property is used like GlobList. It is a space-separated list of MIME Types. The MIMEListType determines how they affect indexing. The following is an example of the value for this property:

```
"text/html text/plain application/*"
```

This property is read-write.

See Also

Chapter 45, "Verity Search Classes," MIMEListType, page 2520 and Chapter 45, "Verity Search Classes," GlobList, page 2518

MIMEListType

Description

This property is used like GlobListType, but applies to MIMEList instead of GlobList. Instead of testing the pattern of the filename, this property tests the MIME-type detected for the document when determining whether to index it.

This property is read-write.

See Also

Chapter 45, "Verity Search Classes," MIMEList, page 2519 and Chapter 45, "Verity Search Classes," GlobListType, page 2519

StartOpts

Description

This property returns a reference to a Search Start Options collection.

This property is read-only.

See Also

Chapter 45, "Verity Search Classes," Search Start Options Collection, page 2520

Search Start Options Collection

The values in the Search Start Options Collection get passed to the −start parameter of the Verity spider utility. These values represent the starting points for spidering.

A Search Start Option collection is returned from:

- StartOpts Search HTTP Options class property
- StartOpts Search FS Options class property

See Search HTTP Options Class property

See Chapter 45, "Verity Search Classes," StartOpts, page 2518.

See Search FS Options Class property

See Chapter 45, "Verity Search Classes," StartOpts, page 2518.

Search Start Options Collection Methods

In this section, we discuss the Search Start Options collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

`DeleteItem``(`*`Value`*`)`

Description

Use the DeleteItem method to delete the Search Start Options specified by *Value.*

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

Parameters

Parameter	***Description***
Value	Specify the value of the Search Start Option that you want to delete.

Returns

A Boolean value: True, item was deleted successfully, False otherwise.

First

Syntax

`First``()`

Description

The First method returns the first Search Start Options object in the Search Start Options collection. If the Search Start Options collection is empty, this method returns Null.

Example

```
&MySearchOpts = &MyCollection.First();
```

InsertItem

Syntax

InsertItem*(Value)*

Description

Use the InsertItem method to insert a Search Start Options object into the Search Start Options collection.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

If the item you are trying to insert already exists, this method returns Null.

Parameters

Parameter	***Description***
Value	Specify the value of the Search Start Option that you want to insert.

Returns

A reference to a new Search Start Option object if successful, Null if not successful.

ItemByName

Syntax

ItemByName*(Value)*

Description

The ItemByName method returns the Search Start Options object specified by *Value*.

Parameters

Parameter	*Description*
Value	Specify the value of the Search Start Options that you want returned.

Returns

A Search Start Options object if successful, Null otherwise.

Example

```
&MySearchOpts = &MyCollection.ItemByName("MyStartOptions");
```

Next

Syntax

Next()

Description

The Next method returns the next Search Start Options object in the Search Start Options collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MySearchOpts = &MyCollection.Next();
```

Search Start Options Collection Property

In this section, we discuss the Search Start Options collection property Count.

Count

Description

This property returns the number of Search Start Options objects in the Search Start Options collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Search Start Options Class

A reference to a Search Start object is returned by the First, ItemByName, InsertItem, and Next Search Start Options Collection methods.

See Chapter 45, "Verity Search Classes," Search Start Options Collection, page 2520.

Search Start Options Class Properties

In this section, we discuss the Search Start Options class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

IsDomainRestricted

Description

The value of this property is passed to the **−domain** option of the Verity spider utility. This property specifies whether the domain is restricted. This property takes a Boolean value: True, the domain is restricted, False otherwise.

This property is read-write.

IsHostRestricted

Description

This property specifies whether the host is restricted. This property takes a Boolean value: True, the host is restricted, False otherwise.

This property is read-write.

Value

Description

This property returns the value of the Search Start Options, as a string.

This property is read-only.

Search API Examples

The following PeopleCode programs are examples of how to use the Search API.

General Purpose Search API Example

The following is a general example uses two search indexes called CATALOG1 and CATALOG2 for the word "bicycle", with the requested fields called "DESCRIPTION" and "PRICE" in the results.

```
&sqQuery = %Session.GetSearchQuery();
&sqQuery.Indexes = "CATALOG1, CATALOG2";
&sqQuery.Language = %Language;
&sqQuery.Querytext = "bicycle";
&sqQuery.SortSpecification = "PRICE desc DESCRIPTION asc";
&sqQuery.RequestedFields = "PRICE, DESCRIPTION";

/* Execute the search, getting the first 20 results. */
&srcResults = &sqQuery.Execute(1, 20);
If &srcResults <> Null And &srcResults.Count > 0 Then

&srCurrent = &srcResults.First();
Repeat
       Local ApiObject &srFieldColl;
&srFieldsColl = &srCurrent.SearchFields;
       Local ApiObject &srField = &srFieldColl.First();
 Repeat
      /* Do something with this field. */
         &srField = &srFieldColl.Next();
       Until None(&srField);
   Until None(&srCurrent);
Else
   ReportSearchAPIErrors();
   Local string &msg;
   &msg = MsgGetText(145, 40, "No results for your search");
End-If;

Function ReportSearchAPIErrors()
   Local ApiObject &msgColl = %Session.PSMessages;
   Local number &i;
   For &i = 1 To &msgColl.count
      Local ApiObject &msg = &msgColl.item(&i);
      Error (&msg.Text);
   End-For;
   &msgColl.DeleteAll();
End-Function;
```

Portal Search API Example

The following is a Portal search example.

```
&SearchKey = %Request.GetParameter("SEARCH_TEXT");
&StartPosition = %Request.GetParameter("START_POSITION")
SearchResultChunkSize = 1000;

/*-------------------------------------------------------------------------------
  Get a portal registry object.
--------------------------------------------------------------------------------*/
&Portal = %Session.GetPortalRegistry();

/*-------------------------------------------------------------------------------
  Open the desired portal.
--------------------------------------------------------------------------------*/
&Portal.Open("PORTAL")

/*------------------------------------------------------------------------------
  Get a search query object.
--------------------------------------------------------------------------------*/
&SearchQuery = &PORTAL.GetSearchQuery();

/*------------------------------------------------------------------------------
  Set the text of the query the user entered
--------------------------------------------------------------------------------*/
&SearchQuery.QueryText = &SearchKey

/*------------------------------------------------------------------------------
  Execute the search passing in the starting position and "chunk size" (get me the⇒
 first 20 result or the third  20)
--------------------------------------------------------------------------------*/
&SearchResultsColl = &SearchQuery.Execute(&StartPosition, &SearchResultChunkSize);
/*------------------------------------------------------------------------------
 Get the first result.
--------------------------------------------------------------------------------*/
&SearchResult = &SearchResultsColl.First();

While (&SearchResult <> Null)

   /*----------------------------------------------------------------------------
     Example of getting the key.  In the portal case, the key is the URL, but in⇒
 the general case it could be a product id or some kind of other database key.
   -----------------------------------------------------------------------------*/
   &SearchKey = &SearchResult.Key;

   /*----------------------------------------------------------------------------
     Example of getting the Field.  In the portal case, the field is the URL. ⇒
 Yes, it⇒
 is redundant with the key but the key includes some {} around the URL and this⇒
 just makes it easier to get the URL.
   -----------------------------------------------------------------------------*/
   &URLSearchField = &SearchResult.SearchFields.ItemByName("URL")
   &URL = &URLSearchField.Value

   /*----------------------------------------------------------------------------
     Another example of a field using dot notation to simplify the code.
   -----------------------------------------------------------------------------*/
   &ContentProvider = &SearchResult.SearchFields.ItemByName("Content⇒
Provider").Value

   /*----------------------------------------------------------------------------
     Call a function to insert the result into the page.
   -----------------------------------------------------------------------------*/
   AddStuffToPage(&URL, &ContentProvider);
```



```
    /*-----------------------------------------------------------------------------
      Get the next result of the search.
    -----------------------------------------------------------------------------*/
    &SearchResult = &SearchResultsColl.Next();

End-While;
&Portal.Close();
```

Chapter 46

XmlDoc Classes

This chapter provides an overview of the XmlDoc class and the XmlNode class and discusses the following topics:

- When to use an XmlDoc object
- XmlDoc object creation
- XmlNode class considerations
- Error handling
- SoapDoc objects consideration
- Scope of XmlDoc and XmlNode objects
- Data type of an XmlDoc or XmlNode object
- XmlDoc classes built-in functions
- XmlDoc class methods
- XmlDoc class property
- XmlNode class methods
- XmlNode class properties

Understanding XmlDoc Classes

The Extended Markup Language (XML) describes a class of data objects called XML documents. It also partially describes the behavior of computer programs which process them. The XmlDoc class is used to create and manipulate XML data.

The Extended Markup Language (XML) is a method for putting structured data in a text file. Like HTML, XML uses tags, that is, text delimited by brackets (< and >). However, HTML specifies what each tag is, and how it's supposed to be displayed in a browser. XML uses tags only to delimit data. The interpretation of that data is entirely up to the application.

For example, <UL> in HTML specifies an unordered (bulleted) list. However, with XML, it could specify an underlined link.

Each XML document has both a physical and a logical structure:

- Physically, the XML document is composed of units called *entities.* A document begins in a "root" or document entity.
- Logically, the XML document is composed of declarations, elements, comments, character references, and processing instructions

The CreateXmlDoc function creates an XmlDoc object. An XmlDoc is composed of XmlNode objects. Each XmlNode represents an XML document *entity.* You can use PeopleCode to create the following types of entities:

- Attribute (specified both by a name and a NamespaceURI)
- CDATA Section
- Comment
- Element
- Entity References
- Process Instruction
- Text

See Also

http://www.w3.org

When to Use an XmlDoc Object

You can use the XmlDoc class to access inbound messages. You can also use the XmlDoc class to create XML messages to be sent out, either synchronously or asynchronously.

All messages must be associated with a message definition. An XmlDoc message must be associated with an *unstructured* or nonrowset-based message, that is, a message definition that's created without any record definitions.

Use the XmlDoc object if any of the following is true:

- Your message structure doesn't fit the rowset model.
- Your message data doesn't come from database records.
- Your third-party source or target node requires non-XML messages.
- Your message uses the SOAP protocol.

Do *not* use the XmlDoc object if your message contains cookies.

XmlDoc Object Creation

Use the CreateXmlDoc function to create an XmlDoc object. You can create either an empty object, and populate it with data, or you can specify an XML string that is then transformed into an XmlDoc object that you can then manipulate with PeopleCode.

If you're creating an empty XmlDoc object, and you also want to specify a particular document type declaration (DTD) to be used to validate your XML, immediately after you use the CreateXmlDoc function, you should use the CreateDocumentType method to specify the DTD, followed by the CreateDocumentElement method to associate the DTD with the root entity.

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &docTypeNode;
Local XmlNode &rootNode;

&inXMLDoc = CreateXmlDoc("");
&docTypeNode = &inXMLDoc.CreateDocumentType("Personal", "", "Personal.dtd");
&rootNode = &inXMLDoc.CreateDocumentElement("root", "", &docTypeNode);
```

The preceding PeopleCode program produces the following XML:

```
<?xml version="1.0"?>
<!DOCTYPE Personal SYSTEM "Personal.dtd">
</root>
```

After you've created your XmlDoc object, use the GenXmlString method to create an XML string. You can then use an Internet Script (iScript) Response object method to send the string as an XML response.

See Also

Chapter 21, "Internet Script Classes (iScript)," page 1051

Considerations Using a Unique Namespace

In the root tag, the attribute **xmlns** stands for the XML namespace. This allows you to define namespaces for tag names so that collisions can be avoided and validation logic can be run.

If you do not give a prefix for an XML namespace, but instead define it with the tag (xmlns) followed by a colon (:) and then a unique namespace, for example,

```
xmlns:psft
```

For example, this sort of functionality allows you to have two nodes named "Transaction", but one can be referenced by a "psft" namespace and another not, allowing for two nodes with the same name to exist, but each containing different data.

```
<?xml version="1.0"?>
<root xmlns:psft="http://www.peoplesoft.com">
    <psft:Transaction>Value</psft:Transaction>
    <Transaction>Another</Transaction>
</root>
```


Incorrect results can be returned when you have a namespace as in the above example, which belongs to the namespace defined in http://www.people.com. When the system tries to find the path of "root/Transaction", it may return multiple nodes when in fact the end user might only want to return one.

To avoid this, do one of the following:

- Do not use FindNode. Use GetElementByTagName instead. This does not use XPath to resolve entries in the DOM. Instead, it works at a node by node basis, for example, by getting the root node, then getting the transactions node. This code may be a bit more complex to write, but you loose the richness of XPath.
- Give every **xmlns** attribute a prefix. The system will use XPath correctly and find the node.

Considerations Using Rowsets

Unstructured XML should be transformed to structured if you want to use the full rowset abilities. PeopleSoft recommends transforming the data into a rowset-based message.

If you do not want to transform the data, you need to break it up using the Transaction tag around the equivalent of each level zero rowset, as shown in the example.


```
<?xml version="1.0" ?>
- <SAMPLE_MSG>
   - <Transaction>
      - <QE_SALES_ORDER class="R">
            <QE_ACCT_ID>26</QE_ACCT_ID>
            <QE_ACCOUNT_NAME>APG-65</QE_ACCOUNT_NAME>
            <QE_ADDRESS>F18 HORNET WAY</QE_ADDRESS>
            <QE_PHONE>(206)544-1264</QE_PHONE>
            <QE_FROMROWSET />
            <QE_TOROWSET />
            <QE_SEND_SOA_BTN />
            <QE_SEND_SOS_BTN />
            <QE_TRAN_SOA_BTN />
            <QE_SEND_SQ_BTN />
            <QE_TRAN_SOS_BTN />
            <QE_TRAN_APCODE_BTN />
            <QE_TRAN_SPCODE_BTN />
            <QE_PUBXMLDOC_BTN />
            <QE_CLEAR_BTN />
         <DESCRLONG />
      </QE_SALES_ORDER>
   </Transaction>
   - <Transaction>
      - <QE_SALES_ORDER class="R">
            <QE_ACCT_ID>27</QE_ACCT_ID>
            <QE_ACCOUNT_NAME>JASON ACCOUNT</QE_ACCOUNT_NAME>
            <QE_ADDRESS>THE ADDRESS</QE_ADDRESS>
            <QE_PHONE>(PHONE NUMBER</QE_PHONE>
            <QE_FROMROWSET />
            <QE_TOROWSET />
            <QE_SEND_SOA_BTN />
            <QE_SEND_SOS_BTN />
            <QE_TRAN_SOA_BTN />
            <QE_SEND_SQ_BTN />
            <QE_TRAN_SOS_BTN />
            <QE_TRAN_APCODE_BTN />
            <QE_TRAN_SPCODE_BTN />
            <QE_PUBXMLDOC_BTN />
            <QE_CLEAR_BTN />
         <DESCRLONG />
      </QE_SALES_ORDER>
   </Transaction>
</SAMPLE_MSG>
```

XmlNode Class Considerations

In general, the XmlNode Class methods and properties can be broken up into the following categories:

Category	***Description***
Add	The Addxxx methods add an entity of the specified type to the end of the list of child nodes, and returns a reference to the newly created node.

Category	*Description*
Insert	The Insertxxx methods insert an entity of the specified type at the specific location and returns a reference to the newly created node.
Get	The Getxxx methods return a reference to the specified entity. The Getxxx methods may return a single reference or an array of references.
Other	The other methods enable you to find a particular entity in an XmlDoc (FindNode), copy data from one node into another, and remove nodes.

Accessing and Traversing an XmlNode Object

If you're creating an XmlDoc object, use the CreateDocumentElement, as well as the different Add and Insert methods to create XmlNode objects. These methods return a reference to the newly created node if successful.

Use the XmlNode properties for traversing the data structure (ParentNode, PreviousSibling, NextSibling, and so on.)

You can also use the ChildNodeCount property for looping through all the child nodes of a node.

Use the NodeType property to get the type of the node (element, processing instruction, comment, and so on).

Error Handling

Use the IsNull property to verify if the XmlNode returned by a method is a valid node. In the following example, a particular node is checked to see if it has another node after it. If it doesn't, a node is added.

```
Local XmlNode &usernode;
Local string &userName;

&usernode = &inXMLDoc.DocumentElement.FindNode("/Request/Company/Location/User");

If Not (&MyNode.IsNull) Then
   &userName = &usernode.NodeValue;
Else
   /* Do error processing */
End-If;
```

SOAPDoc Object Considerations

A SOAP Message is an ordinary XML document that consists of the following parts:

- A mandatory SOAP envelope
- An optional SOAP header
- A mandatory SOAP body

Most of the methods and properties that you use with an XmlDoc or XmlNode can also be used with a SOAPDocument. However, there are a few XmlNode methods that cannot be used with a SOAPDocument. The following methods *cannot* be used with a SOAPDocument.

- AddCDataSection
- AddProcessInstruction
- CopyToPSFTMessage
- CreateDocumentType
- InsertCDataSection
- InsertProcessInstruction
- LoadIBContent

See Also

Chapter 39, "SOAPDoc Class," page 2211

Scope of XmlDoc and XmlNode Objects

XmlDoc and XmlNode objects can be instantiated only from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in Application Engine PeopleCode, record field PeopleCode, and so on.

Data Type of an XmlDoc or XmlNode Object

XmlDoc objects are declared type XmlDoc.

XmlNode objects are declared type XmlNode.

For example:

```
Component XmlDoc &MyDoc;

Local XmlNode &MyNode;
```

Note. XmlNode objects can be declared only as type Local.

XmlDoc Classes Built-in Functions

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSOAPDoc

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateXmlDoc

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetNRXmlDoc

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReValidateNRXmlDoc

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Transform

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TransformEx

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TransformExCache

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," UpdateXmlDoc

XmlDoc Class Methods

In this section, we discuss the XmlDoc methods. The methods are discussed in alphabetical order.

CopyRowset

Syntax

CopyRowset*(&InRowset*[, *MessageName*][, *MessageVersion*])

Description

Use the CopyRowset method to copy rowset data to an XmlDoc object.

Warning! If the XmlDoc is *not* empty, the existing structure and data are replaced with the data and structure from *InRowset* . If *MessageName* is not specified, this function makes the best possible XML structure given the passed rowset. Not passing the message name should only occur when using a nonrowset-based message or when using a standalone rowset. For best performance, PeopleSoft recommends to always specify the message name.

Parameters

Parameter	*Description*
&InRowset	Specify the variable of an already instantiated and populated rowset to copy data from.
MessageName	Specify the name of the message that the rowset belongs to, as a string. If the message name is not specified, the best possible XML structure is created.
MessageVersion	Specify the message version, as a string. If you do not specify a message version, the default message version is used.

Returns

A Boolean value: True, the rowset data copied successfully, False otherwise.

Example

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.CopyRowset(&rs, "<insert message name>", "<insert message⇒
 version");
```

See Also

Chapter 46, "XmlDoc Classes," CopyToRowset, page 2539

Chapter 36, "Rowset Class," page 2127

CopyToPSFTMessage

Syntax

`CopyToPSFTMessage``(`*`targetDoc, srcPath, targetPath`*`)`

Description

Use the CopyToPSFTMessage method to copy an XmlDoc to a PeopleSoft message. The *srcPath* and *targetPath* parameters enable you to map the data of the XmlDoc (using the XmlDoc structure) to the message (following the message structure.)

The XmlDoc object generally has the following structure:


```
<?xml version="1.0"?>
<MESSAGE_NAME>
  <FieldTypes>
    ...
  </FieldTypes>
  <MsgData>
    <Transaction>
      ...
    </Transaction>
  </MsgData>
</MESSAGE_NAME>
```

Parameters

Parameter	***Description***
targetDoc	Specify the XmlDoc that contains the message structure.
srcPath	Specify an array of string that contains the mapping path information.
targetPath	Specify an array of string that contains the mapping path information.

Returns

A number. The values are:

Value	***Description***
0	Ok
1	Missing PSCAMA structure
2	Missing FieldType structure
3	General Error

Example

```
Local XmlDoc &srcDoc, &targetDoc;
Local array of string &srcPath, &targetPath;

&srcPath = CreateArrayRept("", 0);
&targetPath = CreateArrayRept("", 0);


&srcDoc = CreateXmlDoc("");
&targetDoc = CreateXmlDoc("");
&ret = &srcDoc.ParseXmlFromURL("c:/temp/source.xml");
&ret = &targetDoc.ParseXmlFromURL("c:/temp/target.xml");

&srcPath.Push("QE_EMPLOYEE");
&srcPath.Push("QE_EMPLOYEE/QE_ACCOUNT_TBL");
&srcPath.Push("QE_EMPLOYEE/QE_JOBCODE");
&srcPath.Push("QE_EMPLOYEE/DEPTID");
&srcPath.Push("EMAIL_MSG_RCD");
&srcPath.Push("EMAIL_MSG_RCD/EMAIL_FILE_RCD");

&targetPath.Push("QE_EMPLOYEE_TG");
&targetPath.Push("QE_EMPLOYEE_TG/QE_ACCOUNT_TBL_TG");
&targetPath.Push("QE_EMPLOYEE_TG/QE_JOBCODE_TG");
&targetPath.Push("QE_EMPLOYEE_TG/DEPTID_TG");
&targetPath.Push("EMAIL_MSG_RCD_TG");
&targetPath.Push("EMAIL_MSG_RCD/EMAIL_FILE_RCD_TG");

&ret = &srcDoc.CopyToPSFTMessage(&targetDoc, &srcPath, &targetPath);
```

See Also

Chapter 46, "XmlDoc Classes," CopyRowset, page 2536; Chapter 46, "XmlDoc Classes," CopyToRowset, page 2539 and Chapter 8, "Array Class," page 257

Chapter 25, "Message Classes," page 1245

CopyToRowset

Syntax

CopyToRowset*(&Rowset,Message_Name,* [*Message_Version*])

Description

Use the CopyToRowset method to copy data from the XmlDoc to an already instantiated rowset.

The rowset must be based on a message object that has the same *structure* as the XmlDoc. That is, if you have a record at level two in your message and you want that data, you must have the same level zero and level one records in your message as in your XmlDoc.

For example, suppose your XmlDoc had the following structure:


```
<?xml version="1.0"?>
<QE_SALES_ORDER>
    <QE_ACCT_ID/>
    <QE_ACCOUNT_NAME/>
    <QE_ADDRESS/>
    <QE_PHONE/>
    <QE_FROMROWSET/>
    <QE_TOROWSET/>
    <QE_SEND_SOA_BTN/>
    <QE_SEND_SOS_BTN/>
    <QE_TRAN_SOA_BTN/>
    <QE_SEND_SQ_BTN/>
    <QE_TRAN_SOS_BTN/>
    <QE_TRAN_APCODE_BTN/>
    <QE_TRAN_SPCODE_BTN/>
    <QE_PUBXMLDOC_BTN/>
    <QE_CLEAR_BTN/>
    <DESCRLONG/>
</QE_SALES_ORDER>
```

If you wanted to include the information in the QE_SALES_ORDER record, you would have to have *at least* the following record structure in your message:

```
QE_SALES_ORDER
```

Any records or fields that are in the XmlDoc that aren't in the message (and vice-versa) are ignored.

Rowsets should be created using the following pseudo code:

```
Local Message &msg;
Local Rowset &rs;
Local XmlDoc &inXMLDoc;
Local boolean &ret;

&msg = CreateMessage(OPERATION.<insert message name>);
&rs = &msg.GetRowset();
&inXMLDoc = CreateXmlDoc("<insert XML structure here>");
&ret = &inXMLDoc.CopyToRowset(&rs, "<insert message name>", "<insert message⇒
 version>");
```

XmlDoc objects have to follow the Peoplesoft message format:

```
<?xml version="1.0"?>
<PSmessage>
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>xxx</Field1>
                <Field2>yyy</Field2>
                ...
                <Fieldn>nnn</Fieldn>
                <Record2>
                ...
                </Record2>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

Warning! If *MessageName* is not specified, this function makes the best possible flat structure. Not passing the message name should only occur when using a nonrowset-based message or when using a standalone rowset. For best performance, PeopleSoft recommends to always specify the message name.

Parameters

Parameter	*Description*
&InRowset	Specify the variable of an already instantiated rowset to copy data to.
Message_Name	Specify the message name of the message. If the message name is not specified, the best possible flat structure is created.
Message_Version	Specify the message version, as a string. If the message version is not specified, the default message version is used.

Returns

This function always returns a True value, regardless of the success of the operation.

See Also

Chapter 46, "XmlDoc Classes," CopyRowset, page 2536

Chapter 36, "Rowset Class," page 2127

Chapter 25, "Message Classes," page 1245

CreateDocumentElement

Syntax

`CreateDocumentElement``(`*`TagName`*` [, `*`NamespaceURI`*` ] [, `*`&DocType`*`])`

where *NamespaceURI* can have *one* of the following forms:

`URL.`*`URLname`*

OR a string URL, such as

`http://www.peoplesoft.com/`

Description

Use the CreateDocumentElement method to create the root element of the document.

For *&DocType* you must specify an already instantiated document type node using CreateDocumentType.

Parameters

Parameter	*Description*
TagName	Specify the name of the tag to be used for the document element.
NamespaceURI	Specify the URI that contains the Namespaces for the XmlDoc, as a string.
&DocType	Specify the document type node. This must already be instantiated using CreateDocumentType.

Returns

A reference to an XmlNode object representing the root element.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &docTypeNode;
Local XmlNode &rootNode;

&inXMLDoc = CreateXmlDoc("");
&docTypeNode = &inXMLDoc.CreateDocumentType("Personal", "", "Personal.dtd");
&rootNode = &inXMLDoc.CreateDocumentElement("root", "", &docTypeNode);
```

The preceding PeopleCode program produces the following XML:

```
<?xml version="1.0"?>
<!DOCTYPE Personal SYSTEM "Personal.dtd">
<root/>
```

See Also

Chapter 46, "XmlDoc Classes," CreateDocumentType, page 2542

CreateDocumentType

Syntax

CreateDocumentType(*Name*,*PublicID*,*SystemID*)

Description

Use the CreateDocumentType method to create the document type declaration (a document type node) in the XmlDoc.

You can specify only *one* document type node for an XmlDoc.

You need to create a document type node only if you have a document type definition (DTD). You don't need to specify a document type declaration in your XML. Generally, the DTD file is used for validating the XML.

If you do specify this type of node when creating an XmlDoc in PeopleCode, you *must* specify it immediately following the creation statement. Then, you must specify the document type when you create the root node. This means your code should start with the CreateXmlDoc function, creating the XmlDoc object, then use the CreateDocumentType method, followed by the CreateDocumentElement method, creating the root node.

Considerations Using Public and System IDs

You can specify both a public and a system ID. However, if you want to use a system ID *only*, you must specify a Null string ("") for the public ID. To use a public ID *only*, you must specify a Null string ("") for the system ID. If the system ID and public ID are both Null (""), then the document type is considered as system.

To use the Personal.dtd as a system ID, you must use the following code:

```
&DocType = &MyDoc.CreateDocumentType("Personal", "", "Personal.dtd");
```

This produces the XML:

```
<!DOCTYPE Personal SYSTEM "Personal.dtd">
```

To use the Personal.dtd as a public ID, you must use the following code:

```
&DocType = &MyDoc.CreateDocumentType("Personal", "Personal.dtd", "");
```

This produces the XML:

```
<!DOCTYPE Personal PUBLIC "Personal.dtd">
```

Parameters

Parameter	***Description***
Name	Specify the name of the document type declaration as a string.
PublicID	Specify the public ID of the document type declaration as a string.
SystemID	Specify the system ID of the document type declaration as a string.

Returns

A reference to an XmlNode object representing the document type element.

Example

The following example creates a document type declaration using the DTD named "Personal".

```
Local XmlDoc &inXMLDoc;
Local XmlNode &docTypeNode;
Local XmlNode &rootNode;

&inXMLDoc = CreateXmlDoc("");
&docTypeNode = &inXMLDoc.CreateDocumentType("Personal", "", "Personal.dtd");
&rootNode = &inXMLDoc.CreateDocumentElement("root", "", &docTypeNode);
```

This produces the following XML:

```
<?xml version="1.0"?>
<!DOCTYPE Personal SYSTEM "Personal.dtd">
<root/>
```

See Also

Chapter 46, "XmlDoc Classes," CreateDocumentElement, page 2541

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateXmlDoc

GenFormattedXmlString

Syntax

GenFormattedXmlString()

Description

Use the GenFormattedXmlString method to return an XML string representing the XmlDoc object.

The GenFormattedXmlString method produces an XML string with new line characters and indents. If you want an unformatted XML string, that is, with all the data on a single line, use the GenXmlString method instead.

Parameters

None.

Returns

A formatted XML string.

Example

```
Local XmlDoc &inXMLDoc;
Local string &outStr;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><PSMessage/>");
&outStr = &inXMLDoc.GenFormattedXmlString();
```

See Also

Chapter 46, "XmlDoc Classes," GenXmlString, page 2546

GenXmlFile

Syntax

GenXmlFile*(path)*

Description

Use the GenXmlFile method to write the content of the XmlDoc object to a file. The XML output file will be in UTF-8 format, and its elements will be formatted and named exactly the same as the XML output from the GenFormattedXmlString method.

Parameters

Parameter	*Description*
path	Specifies the absolute system path of the destination file.

Returns

A Boolean value: True, if the XML file was saved successfully, False otherwise.

Example

```
Local XmlDoc &xmldocRoot = CreateXmlDoc(" ");
Local XmlNode &nodeRoot = &xmldocRoot.CreateDocumentElement("root");
.
.
.
&result = &xmldocRoot.GenXmlFile("c:\data.xml");
```

See Also

Chapter 46, "XmlDoc Classes," GenFormattedXmlString, page 2544

GenXmlString

Syntax

GenXmlString()

Description

Use the GenXmlString method to return an XML string representing the XmlDoc object.

The GenXmlString method produces an XML string with all the data on a single line. If you want a formatted XML string, that is, with new line characters and indents, use the GenFormattedXmlString method instead.

Parameters

None.

Returns

An XML string.

Example

```
Local XmlDoc &inXMLDoc;
Local string &outStr;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><PSMessage/>");
&outStr = &inXMLDoc.GenXmlString();
```

See Also

Chapter 46, "XmlDoc Classes," GenFormattedXmlString, page 2544

GetElementsByTagName

Syntax

GetElementsByTagName*(TagName)*

Description

Use the GetElementsByTagName method to return an array of XmlNode objects that match the specified tag name.

If you specify an invalid tag name, an array is still returned, however, it has 0 elements in it.

Parameters

Parameter	*Description*
TagName	Specify the tag name that you want to look for.

Returns

An array of XmlNode objects.

Example

Using the following Xml structure:

```
<?xml version="1.0"?>
<PSmessage>
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>one</Field1>
                <Field1>two</Field1>
                <Field1>three</Field1>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

The following code finds all of the *Field1* elements:

```
Local array of XmlNode &field1List;

&field1List = &inXMLDoc.GetElementsByTagName("Field1");

If &field1List.Len = 0 Then
   /* do error processing */
Else
   /* do processing */
End-If;
```

See Also

Chapter 8, "Array Class," page 257

LoadIBContent

Syntax

`LoadIBContent`*`(Content`*`[, `*`RootTagName`*`])`

Description

Use LoadIBContent to load data into an XmlDoc object. This function is primarily used for display purposes, the main use is found in the Integration Broker Monitor for viewing message structures.

When *Content* is an XML string, the XmlDoc contains the DOM representation of the XML.

When *Content* is not an XML string, LoadIBContents generates a DOM wrapper.

Parameters

Parameter	***Description***
Content	Specify the content of the XmlDoc.
RootTagName	Specify the root tag name to be used in the XmlDoc object used as the wrapper, if the data is non-Xml. If a root tag name is not specified, and the data is non-Xml, the contents are not wrapped with the default of root tag name of PSMessage.

Returns

This method returns a Boolean value that is always True.

Example

Using the following data:

```
John Q. Public,1234 West Eastland,925-987-0909
Jane Doe,423 Someplace,234-992-9383
```

The following PeopleCode program specifies a root name:

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;
Local string &inStr;

&inStr = "John Q. Public,1234 West Eastland,925-987-0909" | Char(13) | "Jane⇒
 Doe,423 Someplace,234-992-9383";

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.LoadIBContent(&inStr, "Root");
```

This produces the following:


```
<?xml version="1.0"?>
<Root>
  <data PsNonXml="Yes">
    <![CDATA[John Q. Public,1234 West Eastland,925-987-0909
Jane Doe,423 Someplace,234-992-9383]]>
  </data>
</Root>
```

The following PeopleCode program does not specify a root name:

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;
Local string &inStr;

&inStr = "John Q. Public,1234 West Eastland,925-987-0909" | Char(13) | "Jane⇒
 Doe,423 Someplace,234-992-9383";

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.LoadIBContent(&inStr);
```

This produces the following:

```
<?xml version="1.0"?>
<PSMessage>
  <data PsNonXml="Yes">
    <![CDATA[John Q. Public,1234 West Eastland,925-987-0909
Jane Doe,423 Someplace,234-992-9383]]>
  </data>
</PSMessage>
```

The following is an example of a valid XML document:

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;
Local string &inStr;

&inStr = "<?xml version='1.0'?><root/>";

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.LoadIBContent(&inStr);
```

This produces the following:

```
<?xml version="1.0"?>
<root/>
```

ParseXmlFromURL

Syntax

ParseXmlFromURL*(path* [, *DTDValidation*])

Where *path* can have *one* of the following forms—a string URL, containing the filename and extension:

```
http://www.peoplesoft.com/filename.ext
```

Or an absolute file path, including the filename and extension:

```
c:\directory\filename.ext
```


Note. HTTPS is not a supported protocol for *path.*

Description

Use the ParseXmlFromURL method to convert the XML file located at *path* into an XmlDoc object that you can then manipulate using PeopleCode. The XmlDoc object executing the method is populated with the XML string after it's been converted. Any data already existing in the XmlDoc object is overwritten.

Using this method also does basic validation of the XML string, comparing it to the document type declaration (DTD) if the DOCTYPE for the DTD is provided in the XML string.

Note. PeopleSoft only supports UTF-8 encoding. Therefore, if the input file is encoded, it must be encoded in UTF-8.

Parameters

Parameter	***Description***
path	Specify the URL or file path to the XML that file you want to manipulate. If you specify a string URL, the URL must be contained in quotation marks. **Note.** HTTPS is not a supported protocol for *path.*
DTDValidation	Specify whether to validate a document type definition (DTD.) This parameter takes a Boolean value. If you specify true, the DTD validation occurs if a DTD is provided. If you specify false, and if a DTD is provided, it is ignored and the XML isn't validated against the DTD. The default value for this parameter is false. In the case of application messaging, if a DTD is provided, it's always ignored and the XML isn't validated against the DTD. If the XML cannot be validated against a DTD, an error is thrown saying that there was an XML parse error.

Returns

A Boolean value: True, the XML file converted successfully and was validated, False otherwise. (If the XML file is not valid, the PeopleCode program terminates.)

Example

The following PeopleCode program loads a file from a file path:

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.ParseXmlFromURL("c:\temp\in.xml");
```

The following PeopleCode program loads a file from a URL:


```
Local XmlDoc &inXMLDoc;
Local boolean &ret;

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.ParseXmlFromURL("http://www.peoplesoft.com/xmlfile.xml");
```

See Also

Chapter 46, "XmlDoc Classes," ParseXmlString, page 2551

ParseXmlString

Syntax

`ParseXmlString`*`(XmlString)`*

Description

Use the ParseXmlString method to convert the XML string into an XmlDoc object that you can then manipulate using PeopleCode. The XmlDoc object executing the method is populated with the XML string after it's been converted. Any data already existing in the XmlDoc object is overwritten.

Using this method also does basic validation of the XML string, comparing it to the document type declaration (DTD) if the DOCTYPE for the DTD is provided in the XML string.

Parameters

Parameter	***Description***
XmlString	Specify an XML string that you want to be able to manipulate using PeopleCode.

Returns

A Boolean value: True, the XML string converted successfully and was validated, False otherwise. (If the XML is not valid, the PeopleCode program terminates.)

Example

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;
Local string &str;

&str = "<?xml version='1.0'?><root/>";

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.ParseXmlString(&str);

If &ret Then
   /* do processing */
Else
   /* do error processing */
End-If;
```

See Also

Chapter 46, "XmlDoc Classes," ParseXmlFromURL, page 2549

XmlDoc Properties

In this section, we discuss the XmlDoc properties. The properties are discussed in alphabetical order.

DocumentElement

Description

This property returns a reference to the root element of the document as an XmlNode.

This property is read-only.

See Also

Chapter 46, "XmlDoc Classes," XmlNode Class, page 2553

IsNull

Description

This property returns a Boolean value, indicating whether the doc is a valid object. This property returns True if the XmlDoc object is valid, False otherwise.

This property is read-only.

XmlNode Class

Unlike C++ and Java programming model, PeopleCode interface uses the root object XmlNode to represent all 11 different derived objects. An XmlNode object can be defined only as local.

XmlNode Class Methods

Add methods create a new node of the specific type, append this new node to the child list, and return this newly created node, while Insert methods add the node to the specified position.

In this section, we discuss the XmlNode methods. The methods are discussed in alphabetical order.

AddAttribute

Syntax

```
AddAttribute(Name,Value);
```

Description

Use the AddAttribute method to add an attribute to an XmlNode. A reference to the newly created attribute is returned.

If you specify an attribute name that already exists, the new value you use *replaces* the existing value, and a reference to the attribute is returned.

Attributes added by the AddAttribute method appear in an arbitrary order in the generated XML.

Parameters

Parameter	***Description***
Name	Specify the name of the Attribute that you want to create, as a string.
Value	Specify the value of the Attribute that you want to create, as a string.

Returns

None.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
<myroot>
   <postreqresponse>
      <candidate>
         <user>
            <location scenery="great" density="low" blank="eh?"/>

         </user>
      </candidate>
   </postreqresponse>
</myroot>
```

Here's the PeopleCode that builds it.

```
Local XmlDoc &inXMLDoc;
Local XmlNode &postReqNode;
Local XmlNode &candidatesNode;
Local XmlNode &userNode;
Local XmlNode &locationNode;
Local XmlNode &sceneryAtt;
Local XmlNode &densityAtt;
Local XmlNode &blankAtt;
Local boolean &ret;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");

&postReqNode = &inXMLDoc.DocumentElement.AddElement("postreqresponse");
&candidatesNode = &postReqNode.AddElement("candidates");
&userNode = &candidatesNode.AddElement("user");
&locationNode = &userNode.AddElement("location");

&locationNode.AddAttribute("scenery", "great");
&locationNode.AddAttribute("density", "low");
&locationNode.AddAttribute("blank", "eh?");
```

See Also

Chapter 46, "XmlDoc Classes," IsNull, page 2590

AddAttributeNS

Syntax

AddAttributeNS*(NamespaceURI,AttributeName,Value)*

where *NamespaceURI* can have *one* of the following forms:

URL.*URLname*

OR a string URL, such as

`http://www.peoplesoft.com/`

Description

Use the AddAttributeNS method to add a namespace attribute to an XmlNode. The new attribute is appended to the list of child nodes.

To insert a Namespace attribute at a particular place in the list of nodes, use the InsertAttributeNS method instead.

A reference to the newly created attribute is returned.

If you specify an attribute name that already exists, the new value you use *replaces* the existing value, and a reference to the attribute is returned.

Parameters

Parameter	***Description***
NamespaceURI	Specify the URI that contains the Namespaces for the XmlDoc, as a string.
AttributeName	Specify the name of the Attribute that you want to create, as a string.
Value	Specify the value of the Attribute that you want to create, as a string.

Returns

None.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&childNode.AddAttributeNS("http://www.peoplesoft.com", "scenery", "great");
```

See Also

Chapter 46, "XmlDoc Classes," IsNull, page 2590

AddCDataSection

Syntax

AddCDataSection*(Data)*

Description

Use the AddCDataSection to add a CDATA section to an XmlNode.

CDATA sections may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup.

To insert a CDATA section at a particular place in the list of nodes, use the InsertCDataSection method instead.

A reference to the newly created CDATA section is returned.

Note. You cannot use this method with a SoapDoc object.

Parameters

Parameter	***Description***
Data	Specify the data to be included in the CDATA section as a string.

Returns

A reference to the newly created CDATA section.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;
Local XmlNode &cdataNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&cdataNode = &childNode.AddCDataSection("testing...");
```

The preceding PeopleCode program produces the following:

```
<?xml version="1.0"?>
<myroot>
  <child>
    <![CDATA[testing...]]>
  </child>
</myroot>
```

See Also

Chapter 46, "XmlDoc Classes," InsertCDataSection, page 2576

AddComment

Syntax

AddComment(*Text*)

Description

Use the AddComment method to add a comment to an XmlNode. The new attribute is appended to the list of child nodes.

To insert a comment at a particular place in the list of nodes, use the InsertComment method instead.

A reference to the newly created comment is returned.

Parameters

Parameter	***Description***
Text	Specify the text to be used as the comment, as a string.

Returns

A reference to the newly created comment.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;
Local XmlNode &commentNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&commentNode = &childNode.AddComment("This is an example of a comment.");
```

The preceding PeopleCode program produces the following XML:

```
<?xml version="1.0"?>
<myroot>
  <child>
    <!--This is an example of a comment.-->
  </child>
</myroot>
```

See Also

Chapter 46, "XmlDoc Classes," InsertComment, page 2577

AddElement

Syntax

AddElement*(TagName)*

Description

Use the AddElement method to add an element to the XmlNode. The new element is appended to the list of child nodes.

To insert an element at a particular place in the list of nodes, use the InsertElement method instead.

A reference to the newly created element is returned.

Parameters

Parameter	***Description***
TagName	Specify the tag for the new element that you are adding, as a string.

Returns

A reference to the newly created element.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
```

The preceding PeopleCode program produces the following XML:

```
<?xml version="1.0"?>
<myroot>
  <child/>
</myroot>
```

See Also

Chapter 46, "XmlDoc Classes," InsertElement, page 2579; Chapter 46, "XmlDoc Classes," GetElement, page 2572 and Chapter 46, "XmlDoc Classes," GetElementsByTagName, page 2574

AddElementNS

Syntax

AddElementNS*(NamespaceURI, TagName)*

Where *NamespaceURI* can have *one* of the following forms:

URL.*URLName*

Or a string URL, such as:

`http://www.peoplesoft.com/`

Description

Use AddElementNS to add a namespace element to an XmlNode. The new element is appendced to the list of child nodes.

To insert a Namespace element at a particular place in the list of nodes, use the InsertElementNS method instead. A reference to the newly created element is returned. If you specify an element name that already exists, the new value you use replaces the existing value, and a reference to the element is returned.

Parameters

Parameter	***Description***
NamespaceURI	Specify the URI that contains the Namespaces for the XmlDoc, as a string.
TagName	Specify the name of the Element that you want to create, as a string.

Returns

A reference to the newly created element if successful. If not successful, the IsNull property is set to True.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElementNS("http://www.peoplesoft.com",⇒
 "child");
```

AddEntityReference

Syntax

AddEntityReference*(Name)*

Description

Use the AddEntityReference method to add an entity reference. An entity reference refers to the content of the named entity. The new entity reference is appended to the list of child nodes. In the generated XML, the entity reference is automatically prefaced with the '&' character and suffixed with a semi-colon.

The named entity must exist in the document type declaration (DTD).

To insert an entity reference at a particular place in the list of nodes, use the InsertEntityReference method instead.

Parameters

Parameter	***Description***
Name	Specify the name of the entity to which this reference refers to.

Returns

A reference to the newly created entity reference.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;
Local XmlNode &entityRef;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&entityRef = &childNode.AddEntityReference("MyURL");
```

The preceding PeopleCode program produces the following XML:

```
<?xml version="1.0"?>
<myroot>
  <child>
&MyURL;  </child>
</myroot>
```


See Also

Chapter 46, "XmlDoc Classes," InsertEntityReference, page 2581

AddNode

Syntax

AddNode*(&XmlNode)*

Description

Use the AddNode method to add an XmlNode to the XmlDoc. The new node is appended to the list of child nodes.

To insert a node at a particular place in the list of nodes, use the InsertNode method instead.

A reference to the newly created node is returned.

Parameters

Parameter	***Description***
&XmlNode	Specify an already instantiated XmlNode that you want to add.

Returns

None.

Example

```
Local XmlDoc &inXMLDoc, &firstDoc;
Local XmlNode &childNode;

&firstDoc = CreateXmlDoc("<?xml version='1.0'?><myroot><child><subchild/></child><⇒
/myroot>");
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&childNode = &firstDoc.DocumentElement.FindNode("/child");
&inXMLDoc.DocumentElement.AddNode(&childNode);
```

See Also

Chapter 46, "XmlDoc Classes," InsertNode, page 2582

AddProcessInstruction

Syntax

`AddProcessInstruction`*`(Target,Data)`*

Description

Use the AddProcessInstruction method to add a processing instruction to an XmlNode. The new process instruction is appended to the end of the child list.

To insert a processing instruction at a particular place, use the InsertProcessInstruction method instead.

A reference to the newly created processing instruction is returned.

Note. You cannot use this method with a SoapDoc object.

Parameters

Parameter	*Description*
Target	Specify the application to which the instruction is directed, as a string.
Data	Specify the data used with the processing instruction.

Returns

A reference to the newly created processing instruction.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &procInst;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&procInst = &inXMLDoc.DocumentElement.AddProcessInstruction("xml-stylesheet",⇒
 "href=""book.css"" type=""text/css""");
```

The preceding PeopleCode program produces the followingXML :

```
<?xml version="1.0"?>
<myroot>
  <?xml-stylesheet href="book.css" type="text/css"?>
</myroot>
```


See Also

Chapter 46, "XmlDoc Classes," InsertProcessInstruction, page 2584

AddText

Syntax

AddText*(Data)*

Description

Use the AddText method to add a text node to an XmlNode.

Note. A text node is *not* the same as a text declaration. The text declaration is added automatically with the CreateXmlDoc function. A text node just contains text within an XmlDoc.

To insert a text node at a particular place, use the InsertText method instead.

A reference to the newly created text node is returned.

Parameters

Parameter	***Description***
Data	Specify the data to be used as the text node, as a string.

Returns

A reference to the newly created text node.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;
Local XmlNode &textNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&textNode = &childNode.AddText("This is text");
```

The preceding PeopleCode program produces the following XML:


```
<?xml version="1.0"?>
<myroot>
  <child>This is text</child>
</myroot>
```

See Also

Chapter 46, "XmlDoc Classes," InsertText, page 2585

CopyNode

Syntax

CopyNode*(&Node)*

Description

Use the CopyNode method to copy a node specified by *&Node* and *all* of its child nodes into the current node, as a child node. The new node is appended to the end of the list of child nodes. The specified node can belong to a different XmlDoc.

If you just want to copy a top-level node, without copying its children, you can just create a new node with the name of the node you want.

Parameters

Parameter	***Description***
&Node	Specify an already instantiated node that you want to copy.

Returns

None.

Example

Suppose that your XmlDoc has the following structure:

```
<?xml version="1.0"?>
<myroot>
  <child>
    <subchild/>
  </child>
</myroot>
```

The following PeopleCode copies the node *child* and copies it into another doc.


```
Local XmlDoc &inXMLDoc, &firstDoc;
Local XmlNode &childNode;

&firstDoc = CreateXmlDoc("<?xml version='1.0'?><myroot><child><subchild/></child><⇒
/myroot>");
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&childNode = &firstDoc.DocumentElement.FindNode("/child");
&inXMLDoc.DocumentElement.CopyNode(&childNode);
```

The following is the new structure:

```
<?xml version="1.0"?>
<root>
  <child>
    <subchild/>
  </child>
</root>
```

See Also

Chapter 46, "XmlDoc Classes," AddNode, page 2561; Chapter 46, "XmlDoc Classes," InsertNode, page 2582; Chapter 46, "XmlDoc Classes," GetChildNode, page 2571; Chapter 46, "XmlDoc Classes," FindNode, page 2565 and Chapter 46, "XmlDoc Classes," FindNodes, page 2566

FindNode

Syntax

FindNode*(Path)*

Description

Use the FindNode method to return a reference to an XmlNode.

The path is specified as the list of tag names, to the node that you want to find, each separated by a slash (/).

Parameters

Parameter	***Description***
Path	Specify the tag names up to and including the name of the node that you want returned, starting with a slash and each separated by a slash (/). This is known as the XPath query language. See http://www.w3.org/TR/xpath.

Returns

An XmlNode matching the path if successful. If not successful, the IsNull property on the XmlNode is set to True.

Example

Suppose your XmlDoc has the following structure:

```
<?xml version="1.0"?>
<myroot>
   <postreqresponse>
      <candidate>
         <user>
            <location scenery="great" density="low" blank="eh?"/>
         </user>
      </candidate>
   </postreqresponse>
</myroot>
```

You want to return a reference to the location attribute. Use the following *Path* to do it:

```
&XmlNode = &MyDoc.FindNode("/myroot/postreqresponse/candidate/user/location");
```

See Also

Chapter 46, "XmlDoc Classes," GetChildNode, page 2571 and Chapter 46, "XmlDoc Classes," FindNodes, page 2566

FindNodes

Syntax

FindNodes*(Path)*

Description

Use the FindNodes method to return a reference to an array containing all the XmlNodes that match the path.

The path is specified as the list of tag names, to the node that you want to find, each separated by a slash (/).

Parameters

Parameter	*Description*
Path	Specify the tag names up to and including the name of the node that you want returned, each separated by a slash (/). This is known as the XPath query language. See http://www.w3.org/TR/xpath.

Returns

An array of XmlNode objects. If there is no match for the specified path, an array with 0 elements is returned.

Example

Using the following input:

```
<?xml version="1.0"?>
<PSmessage>
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>one</Field1>
                <Field1>two</Field1>
                <Field1>three</Field1>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

The following PeopleCode program finds all of the *Field1* nodes:

```
Local array of XmlNode &field1List;

&field1List = &inXMLDoc.DocumentElement.FindNodes("MsgData/Transaction/Record1⇒
/Field1");

If &field1List.Len = 0 Then
   /* do error processing, no nodes returned */
Else
   /* do regular processing */
End-If;
```

See Also

Chapter 46, "XmlDoc Classes," GetChildNode, page 2571; Chapter 46, "XmlDoc Classes," FindNode, page 2565 and Chapter 8, "Array Class," page 257

GenXmlString

Syntax

GenXmlString()

Description

Use the GenXmlString method to generate an XML string of the XmlNode.

Parameters

None.

Returns

An XML String.

Example

```
&MyXML = &MyNode.GenXmlString();
```

See Also

Chapter 46, "XmlDoc Classes," GenXmlString, page 2567 XmlDoc method

GetAttributeName

Syntax

GetAttributeName*(Index)*

Description

Use the GetAttributeName method to return the specified attribute.

Parameters

Parameter	***Description***
Index	Specify an integer representing the attribute you want to access.

Returns

A string.

Example

```
Local XmlDoc &inXMLDoc;
Local string &attName;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&inXMLDoc.DocumentElement.AddAttribute("name", "Joe");
&inXMLDoc.DocumentElement.AddAttribute("address", "1234 West Eastland");

&attName = &inXMLDoc.DocumentElement.GetAttributeName(2);
```


GetAttributeValue

Syntax

GetAttributeValue({*Name* | *Index*})

Description

Use the GetAttributeValue method to return the specific attribute value, given an attribute name. The attribute name is case-sensitive, so you must specify the exact name.

Parameters

Parameter	***Description***
Name | *Index*	Specify the name or index of the attribute whose value you want to access.

Returns

A string containing the value of the specified attribute.

Example

```
Local XmlDoc &inXMLDoc;
Local string &attName;
Local string &attValue;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&inXMLDoc.DocumentElement.AddAttribute("name", "Joe");
&inXMLDoc.DocumentElement.AddAttribute("address", "1234 West Eastland");

&attName = &inXMLDoc.DocumentElement.GetAttributeName(2);
&attValue = &inXMLDoc.DocumentElement.GetAttributeValue(&attName);
```

GetCDataValue

Syntax

GetCDataValue()

Description

Use the GetCDataValue method to return the *value* of the first CDATA section in an XmlNode.

Parameters

None.

Returns

A reference to the value of the first CDATA section as a string.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &cdataNode;
Local string &theData;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&cdataNode = &dataNode.AddCDataSection("this is a bunch of text");

&theData = &dataNode.GetCDataValue();
```

See Also

Chapter 46, "XmlDoc Classes," AddCDataSection, page 2555; Chapter 46, "XmlDoc Classes," InsertCDataSection, page 2576 and Chapter 46, "XmlDoc Classes," GetCDataValues, page 2570

GetCDataValues

Syntax

GetCDataValues()

Description

Use the GetCDataValues method to return an array of string containing the *values* of the CDATA section in an XmlNode.

Parameters

None.

Returns

An array of string.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &cdataNode;
Local array of string &theData;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&cdataNode = &dataNode.AddCDataSection("this is a bunch of text");
&cdataNode = &dataNode.AddCDataSection("more text");
&cdataNode = &dataNode.AddCDataSection("still more text");

&theData = &dataNode.GetCDataValues();
```

See Also

Chapter 46, "XmlDoc Classes," AddCDataSection, page 2555; Chapter 46, "XmlDoc Classes," InsertCDataSection, page 2576; Chapter 46, "XmlDoc Classes," GetCDataValue, page 2569 and Chapter 8, "Array Class," page 257

GetChildNode

Syntax

GetChildNode*(index)*

Description

Use the GetChildNode method to return the specified child node of an XmlNode.

Parameters

Parameter	***Description***
Index	Specify an integer representing the child node that you want to access.

Returns

A reference to an XmlNode. If the specified XmlNode isn't found, the IsNull property for the XmlNode is set to True.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data2");

&dataNode = &inXMLDoc.DocumentElement.GetChildNode(2);
```

See Also

Chapter 46, "XmlDoc Classes," FindNode, page 2565 and Chapter 46, "XmlDoc Classes," FindNodes, page 2566

GetElement

Syntax

GetElement()

Description

Use the GetElement method to return the first child element node in the list of child nodes.

Parameters

None.

Returns

A reference to an element.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data2");

&dataNode = &inXMLDoc.DocumentElement.GetElement();
```


See Also

Chapter 46, "XmlDoc Classes," AddElement, page 2558; Chapter 46, "XmlDoc Classes," InsertElement, page 2579; Chapter 46, "XmlDoc Classes," GetElements, page 2573 and Chapter 46, "XmlDoc Classes," GetElementsByTagName, page 2574

GetElements

Syntax

GetElements()

Description

Use the GetElements method to return an array of all the XmlNode objects that are elements.

Parameters

None.

Returns

An array of XmlNode of all the element nodes. If there are no element nodes, an array with 0 elements is returned.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local array of XmlNode &dataList;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data2");

&dataList = &inXMLDoc.DocumentElement.GetElements();

If &dataList.Len = 0 Then
   /* do error processing, no nodes returned */
Else
   /* do regular processing */
End-If;
```

See Also

Chapter 46, "XmlDoc Classes," AddElement, page 2558; Chapter 46, "XmlDoc Classes," InsertElement, page 2579; Chapter 46, "XmlDoc Classes," GetElement, page 2572; Chapter 46, "XmlDoc Classes," GetElementsByTagName, page 2574 and Chapter 8, "Array Class," page 257

GetElementsByTagName

Syntax

`GetElementsByTagName`*`(TagName)`*

Description

Use the GetElementsByTagName method to return an array of XmlNode objects that match the specified tag name.

Parameters

Parameter	*Description*
TagName	Specify the tag name that you want to look for.

Returns

An array of XmlNode. If you specify an invalid tag name, the returned array has 0 elements.

Example

Using the following input:

```
<?xml version="1.0"?>
<PSmessage>
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>one</Field1>
                <Field1>two</Field1>
                <Field1>three</Field1>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

The following PeopleCode program finds all *Field1* nodes:

```
Local array of XmlNode &field1List;

&field1List = &inXMLDoc.DocumentElement.GetElementsByTagName("Field1");

If &field1List.Len = 0 Then
   /* do error processing, no nodes returned */
Else
   /* do regular processing */
End-If;
```


See Also

Chapter 46, "XmlDoc Classes," AddElement, page 2558; Chapter 46, "XmlDoc Classes," InsertElement, page 2579; Chapter 46, "XmlDoc Classes," GetElements, page 2573; Chapter 46, "XmlDoc Classes," GetElement, page 2572 and Chapter 8, "Array Class," page 257

GetElementsByTagNameNS

Syntax

GetElementsByTagNameNS*(NamespaceURI, TagName)*

Where *NamespaceURI* can have *one* of the following forms:

URL.*URLName*

Or a string URL, such as:

`http://www.peoplesoft.com/`

Description

Use the GetElementsByTagNameNS method to return an array of XmlNode objects that match the specified namespace and qualified name.

Parameters

Parameter	***Description***
NamespaceURI	Specify the URI that contains the Namespace for the XmlDoc, as a string.
TagName	Specify the name of the element that you want to retrieve.

Returns

An array of XmlNode. If you specify an invalid NamespaceURI, the returned array has 0 elements.

Example

Using the following input:


```
<?xml version="1.0"?>
<Psmessage xmlns="http://www.peoplesoft.com">
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>one</Field1>
                <Field1>two</Field1>
                <Field1>three</Field1>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

The following PeopleCode program finds all *Field1* nodes:

```
Local array of XmlNode &field1List;

&field1List = &inXMLDoc.DocumentElement.GetElementsByTagNameNS("http:⇒
//www.peoplesoft.com", "Field1");

If &field1List.Len = 0 Then
   /* do error processing, no nodes returned */
Else
   /* do regular processing */
End-If;
```

InsertCDataSection

Syntax

InsertCDataSection(*Data*,*Position*)

Description

Use the InsertCDataSection method to insert a CDATA section in an XmlNode, at the location specified by *Position.*

Note. You cannot use this method with a SoapDoc object.

Parameters

Parameter	*Description*
Data	Specify the data for the CDATA section as a string.
Position	Specify where you want to insert the CDATA Section, as a number.

Returns

A reference to the newly created CDATA section.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &cdataNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&cdataNode = &dataNode.AddCDataSection("this is a bunch of text");
&cdataNode = &dataNode.AddCDataSection("still more text");

&cdataNode = &dataNode.InsertCDataSection("more text", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <![CDATA[this is a bunch of text]]>
    <![CDATA[still more text]]>
  </data>
</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <![CDATA[this is a bunch of text]]>
    <![CDATA[more text]]>
    <![CDATA[still more text]]>
  </data>
</root>
```

See Also

Chapter 46, "XmlDoc Classes," AddCDataSection, page 2555; Chapter 46, "XmlDoc Classes," GetCDataValue, page 2569 and Chapter 46, "XmlDoc Classes," GetCDataValues, page 2570

InsertComment

Syntax

InsertComment*(Data,Position)*

Description

Use the InsertComment method to insert a comment in an XmlNode at the location specified by *Position.*

Parameters

Parameter	*Description*
Data	Specify the data for the comment as a string.
Position	Specify where you want to insert the comment, as a number.

Returns

A reference to the newly created comment.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &commentNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&commentNode = &dataNode.AddComment("this is a comment");
&commentNode = &dataNode.AddComment("still another comment");

&commentNode = &dataNode.InsertComment("more comments", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <!--this is a comment-->
    <!--still another comment-->
  </data>
</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <!--this is a comment-->
    <!--more comments-->
    <!--still another comment-->
  </data>
</root>
```

See Also

Chapter 46, "XmlDoc Classes," AddComment, page 2557

InsertElement

Syntax

InsertElement*(TagName,Position)*

Description

Use the InsertElement method to insert an element in an XmlNode at the location specified by *Position.*

Parameters

Parameter	*Description*
TagName	Specify the tagname for the element as a string.
Position	Specify where you want to insert the element, as a number.

Returns

A reference to the newly created element.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &elementNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&elementNode = &dataNode.AddElement("first");
&elementNode = &dataNode.AddElement("second");

&commentNode = &dataNode.InsertElement("third", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <first/>
    <second/>
  </data>
</root>
```

This is the XML document after the insert:


```
<?xml version="1.0"?>
<root>
  <data>
    <first/>
    <third/>
    <second/>
  </data>
</root>
```

See Also

Chapter 46, "XmlDoc Classes," AddElement, page 2558

InsertElementNS

Syntax

InsertElementNS*(NamespaceURI,TagName,Position)*

where *NamespaceURI* can have *one* of the following forms:

URL.*URLname*

OR a string URL, such as

`http://www.peoplesoft.com/`

Description

Use InsertElementNS to insert a new element in an XmlNode, at the location specified by *Position.*

Parameters

Parameter	***Description***
NamespaceURI	Specify the URI that contains the Namespaces for the XmlDoc.
TagName	Specify the name of the element that you want to create, as a string.
Position	Specify where you want to insert the element, as a number.

Returns

A reference to the newly created element.

Example

For example:


```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &elementNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root xmlns='http:⇒
//www.peoplesoft.com'/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&elementNode = &dataNode.AddElement("first");
&elementNode = &dataNode.AddElement("second");

&elementNode = &dataNode.InsertElementNS("http://www.peoplesoft.com", "third", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root xmlns="http://www.peoplesoft.com">
  <data>
    <first/>
    <second/>
  </data>
</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root xmlns="http://www.peoplesoft.com">
  <data>
    <first/>
    <third/>
    <second/>
  </data>
</root>
```

InsertEntityReference

Syntax

InsertEntityReference*(Name,Position)*

Description

Use the InsertEntityReference method to add an entity reference. An entity reference refers to the content of the named entity.

Parameters

Parameter	***Description***
Name	Specify the name of the entity to which this reference refers to.
Position	Specify where you want to insert the entity reference, as a number.

Returns

A reference to the newly created entity reference.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &entityRef;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&entityRef = &dataNode.AddEntityReference("first");
&entityRef = &dataNode.AddEntityReference("second");

&entityRef = &dataNode.InsertEntityReference("third", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <data>
&first;&second;  </data>
</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
  <data>
&first;&third;&second;  </data>
</root>
```

See Also

Chapter 46, "XmlDoc Classes," AddEntityReference, page 2560

InsertNode

Syntax

InsertNode*(&NewNode,* {*&RefNode* | *Position*})

Description

Use the InsertNode method to insert a new node. The node is inserted either before the node specified by *&RefNode,* or at the location specified by *Position.*

&NewNode refers to an already instantiated XmlNode object. You must create the node before you can insert it.

&RefNode refers to an already instantiated XmlNode object.

Parameters

Parameter	***Description***
&NewNode	Specify the name of an already instantiated XmlNode that you want to insert.
&RefNode | *Position*	Specify either the name of an existing node or the location where you want the node to be inserted.

Returns

A reference to the newly inserted node.

Example

The following PeopleCode program inserts a node using a position:

```
Local XmlDoc &inXMLDoc, &firstDoc;
Local XmlNode &childNode;

&firstDoc = CreateXmlDoc("<?xml version='1.0'?><myroot><third><child/></third><⇒
/myroot>");
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root><first/><second/></root>");
&childNode = &firstDoc.DocumentElement.FindNode("/third");

&inXMLDoc.DocumentElement.InsertNode(&childNode, 2);
```

The following PeopleCode program inserts a node using a reference node:

```
Local XmlDoc &inXMLDoc, &firstDoc;
Local XmlNode &childNode, &secondNode;

&firstDoc = CreateXmlDoc("<?xml version='1.0'?><myroot><third><child/></third><⇒
/myroot>");
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root><first/><second/></root>");
&childNode = &firstDoc.DocumentElement.FindNode("/third");
&secondNode = &inXMLDoc.DocumentElement.FindNode("/second");

&inXMLDoc.DocumentElement.InsertNode(&childNode, &secondNode);
```

The following is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <first/>
  <second/>
</root>
```

The following is the XML document after the insert:


```
<?xml version="1.0"?>
<root>
  <first/>
  <third>
    <child/>
  </third>
  <second/>
</root>
```

See Also

Chapter 46, "XmlDoc Classes," AddNode, page 2561

InsertProcessInstruction

Syntax

InsertProcessInstruction*(Target,Data,Position)*

Description

Use the InsertProcessInstruction method to insert a processing instruction to an XmlNode at the location specified by *Position.*

Note. Youcannot use this method with a SoapDoc object.

Parameters

Parameter	*Description*
Target	Specify the application to which the instruction is directed, as a string.
Data	Specify the data to be used with the instruction.
Position	Specify where you want to insert the process instruction, as a number.

Returns

A reference to the newly created processing instruction.

Example

For example:


```
Local XmlDoc &inXMLDoc;
Local XmlNode &procInst;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&procInst = &inXMLDoc.DocumentElement.AddProcessInstruction("first", "firstvalue");
&procInst = &inXMLDoc.DocumentElement.AddProcessInstruction("second",⇒
 "secondvalue");

&procInst = &inXMLDoc.DocumentElement.InsertProcessInstruction("third",⇒
 "thirdvalue", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <?first firstvalue?>
  <?second secondvalue?>
</root>
```

This is the XML document after the inser:

```
<?xml version="1.0"?>
<root>
  <?first firstvalue?>
  <?third thirdvalue?>
  <?second secondvalue?>
</root>
```

See Also

Chapter 46, "XmlDoc Classes," AddProcessInstruction, page 2562

InsertText

Syntax

InsertText(*Data*,*Position*)

Description

Use the InsertText method to insert a text node. The node is inserted at the location indicated by *Position.*

Note. A text node is *not* the same as a text declaration. The text declaration is added automatically with the CreateXmlDoc function. A text node just contains text within an XmlDoc.

Parameters

Parameter	***Description***
Data	Specify the data to be used for the text node, as a string.

Parameter	***Description***
Position	Specify where you want to insert the text ndoe, as a number.

Returns

A reference to the newly created text node.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &textNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&textNode = &inXMLDoc.DocumentElement.AddText("first text");
&textNode = &inXMLDoc.DocumentElement.AddText("second text");

&textNode = &inXMLDoc.DocumentElement.InsertText("third text", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
first textsecond text</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
first textthird textsecond text</root>
```

See Also

Chapter 46, "XmlDoc Classes," AddText, page 2563

RemoveAllChildNode

Syntax

```
RemoveAllChildNode()
```

Description

Use the RemoveAllChildNode method to remove all child nodes for an XmlNode.

Parameters

None.

Returns

None.

Example

For example:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &elementNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&elementNode = &inXMLDoc.DocumentElement.AddElement("first");
&elementNode = &inXMLDoc.DocumentElement.AddElement("second");
&elementNode = &inXMLDoc.DocumentElement.AddElement("third");

&inXMLDoc.DocumentElement.RemoveAllChildNode();
```

This is the XML document before the removal:

```
<?xml version="1.0"?>
<root>
  <first/>
  <second/>
  <third/>
</root>
```

This is the XML document after the removal:

```
<?xml version="1.0"?>
<root/>
```

See Also

Chapter 46, "XmlDoc Classes," AddNode, page 2561; Chapter 46, "XmlDoc Classes," InsertNode, page 2582 ; Chapter 46, "XmlDoc Classes," GetChildNode, page 2571; Chapter 46, "XmlDoc Classes," FindNode, page 2565; Chapter 46, "XmlDoc Classes," FindNodes, page 2566 and Chapter 46, "XmlDoc Classes," RemoveChildNode, page 2587

RemoveChildNode

Syntax

RemoveChildNode({*Position* | *Node*})

Description

Use the RemoveChildNode method to remove the child node in an XmlNode specified by *Node*, or located at *Position.*

Parameters

Parameter	*Description*
Position | Node	Specify the child node that you want to delete, either using its position number or its name.

Returns

A reference to the removed XmlNode.

Example

The following PeopleCode program uses a position:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &elementNode, &removedNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&elementNode = &inXMLDoc.DocumentElement.AddElement("first");
&elementNode = &inXMLDoc.DocumentElement.AddElement("second");
&elementNode = &inXMLDoc.DocumentElement.AddElement("third");

&removedNode = &inXMLDoc.DocumentElement.RemoveChildNode(2);
```

The following PeopleCode program uses a reference node:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &elementNode, &removedNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&elementNode = &inXMLDoc.DocumentElement.AddElement("first");
&elementNode = &inXMLDoc.DocumentElement.AddElement("second");
&elementNode = &inXMLDoc.DocumentElement.AddElement("third");

&elementNode = &inXMLDoc.DocumentElement.FindNode("/second");
&removedNode = &inXMLDoc.DocumentElement.RemoveChildNode(&elementNode);
```

This is the XML document before the removal:

```
<?xml version="1.0"?>
<root>
  <first/>
  <second/>
  <third/>
</root>
```

This is XML document after the removal:

```
<?xml version="1.0"?>
<root>
  <first/>
  <third/>
</root>
```

See Also

Chapter 46, "XmlDoc Classes," AddNode, page 2561; Chapter 46, "XmlDoc Classes," InsertNode, page 2582 ; Chapter 46, "XmlDoc Classes," GetChildNode, page 2571; Chapter 46, "XmlDoc Classes," FindNode, page 2565; Chapter 46, "XmlDoc Classes," FindNodes, page 2566 and Chapter 46, "XmlDoc Classes," RemoveAllChildNode, page 2586

XmlNode Class Properties

The following are the XmlNode properties.

AttributesCount

Description

This property returns the number of attributes for an XmlNode, as a number.

This property is read-only.

Example

```
Local XmlDoc &inXMLDoc;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&inXMLDoc.DocumentElement.AddAttribute("first", "John");
&inXMLDoc.DocumentElement.AddAttribute("last", "Public");
&inXMLDoc.DocumentElement.AddAttribute("address", "1234 West Eastland");

For &i = 1 To &inXMLDoc.DocumentElement.AttributesCount
   /* do some processing of the attributes here */
End-For;
```

ChildNodeCount

Description

This property returns the number of child nodes for an XmlNode, as a number.

This property is read-only.

Index

Description

This property returns the location (or position) of a node in the parent's child list, as a number. The root node always returns a 1.

This property is read-only.

IsNull

Description

This property returns a Boolean value, indicating whether the node is a valid node. Use this property to verify the value of the add or insert XmlNode methods.

This property is read-only.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &elementNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&elementNode = &inXMLDoc.DocumentElement.AddElement("first");

&elementNode = &inXMLDoc.DocumentElement.FindNode("/second");

If &elementNode.IsNull Then
   /* do some error processing here, will not find "second" */
End-If;
```

LocalName

Description

This property returns the local part of a namespace as a string. The local part is the second part of the namespace entry. If you want access to the first part of a namespace, use the Prefix property.

This property is read-only.

See Also

Chapter 46, "XmlDoc Classes," Prefix, page 2593

NamespaceURI

Description

This property returns a reference to the namespace URI as a string.

This property is read-only.

NextSibling

Description

This property returns the next child node in an XmlNode, as an XmlNode.

Use the IsNull property to verify that the XmlNode that is returned is valid.

This property is read-only.

See Also

Chapter 46, "XmlDoc Classes," IsNull, page 2590

NodeName

Description

This property returns the name of the XmlNode as a string. You can use this property to rename an XmlNode.

This property is read-write.

NodePath

Description

This property returns the path of the XmlNode, from the root.

This property is read-only.

NodeType

Description

This property returns the type of the XmlNode. You can use either a numeric value or a constant. The values are:

Numeric Value	***Constant Value***	***Description***
1	%ElementNode	Element node
2	%AttributeNode	Attribute node
3	%TextNode	Text node
4	%CDataSectionNode	CData section node
5	%EntityReferenceNode	Entity reference node
6	%EntityNode	Entity node
7	%ProcessingInstructionNode	Processing instruction node
8	%CommentNode	Comment node
9	%DocumentNode	Document node
10	%DocumentTypeNode	Document type node
11	%NotationNode	Notation node
12	%XmlDeclNode	XML DECL node

NodeValue

Description

This property returns the value of the XmlNode, as a string.

This property is read-write.

ParentNode

Description

This property returns a reference to the parent node for the XmlNode, as an XmlNode object.

Use the IsNull property to verify that the XmlNode that is returned is valid.

This property is read-only.

See Also

Chapter 46, "XmlDoc Classes," IsNull, page 2590

Prefix

Description

This property returns the prefix part of a namespace as a string. The prefix is the first part of the namespace entry. If you want access to the second part of a namespace, use the LocalName property.

This property is read-only.

See Also

Chapter 46, "XmlDoc Classes," LocalName, page 2590

PreviousSibling

Description

This property returns a reference to the previous node in the XmlNode, as an XmlNode object.

Use the IsNull property to verify that the XmlNode that is returned is valid.

This property is read-only.

See Also

Chapter 46, "XmlDoc Classes," IsNull, page 2590

Chapter 47

XML Publisher Classes

This chapter provides an overview of Oracle's XML Publisher and XML Publisher classes, and discusses the following topics:

- XML Publisher terms.
- XML Publisher flow diagram.
- Error handling.
- Data type and scope of XML Publisher objects.
- Instantiating XML Publisher objects.
- Search operator values.
- XML Publisher classes reference.
- XML Publisher classes example.

See Also

Enterprise PeopleTools 8.51 PeopleBook: XML Publisher for PeopleSoft Enterprise, "Getting Started with XML Publisher"

Understanding XML Publisher and the XML Publisher Classes

Oracle's XML Publisher enables you to separate the data, layout, and translation layers of a report from each other. This can improve flexibility, as well as reduce maintenance. You need to create report definitions, define templates, and so on, using the XML publisher. The XML Publisher classes enable you to access the runtime portions of the XML publishing process programmatically—that is, after the templates and reports have been created.

The XML Publisher classes are divided into the following categories:

- Report manager definition classes
- Report manager search classes
- Engine classes

XML Publisher Report Manager Definition Classes

To create a report with XML Publisher, you first create a report definition using the Report Definition page. The report definition includes report properties, output formats, templates to be used, report security, and so on. At run time, the ReportDefn class uses these attributes to process the report. The ReportDefn class itself can be invoked from a batch process or a page PeopleCode. For example, the report can be scheduled from the Query Report Scheduler page, or viewed from Query Report Viewer page.

See Chapter 47, "XML Publisher Classes," ReportDefn Class, page 2600.

XML Publisher Report Manager Search Classes

After you've published a report, it is stored in the report repository. Using the report manager search classes, you can search for a report based on defined search keys, or even add additional search keys for searching.

See Chapter 47, "XML Publisher Classes," Report Class, page 2613.

See Chapter 47, "XML Publisher Classes," ReportManager Class, page 2618.

See Chapter 47, "XML Publisher Classes," SearchAttribute Class, page 2625.

XML Publisher Engine Classes

Use the XML Publisher engine classes to combine PDF files into a single PDF file, and to specify page numbers and watermarks on merged PDF reports.

See Chapter 47, "XML Publisher Classes," PageNumber Class, page 2627.

See Chapter 47, "XML Publisher Classes," PDFMerger Class, page 2631.

See Chapter 47, "XML Publisher Classes," Properties Class, page 2635.

See Chapter 47, "XML Publisher Classes," Watermark Class, page 2638.

XML Publisher Terms

The following is a list of general XML publisher terms and their definitions.

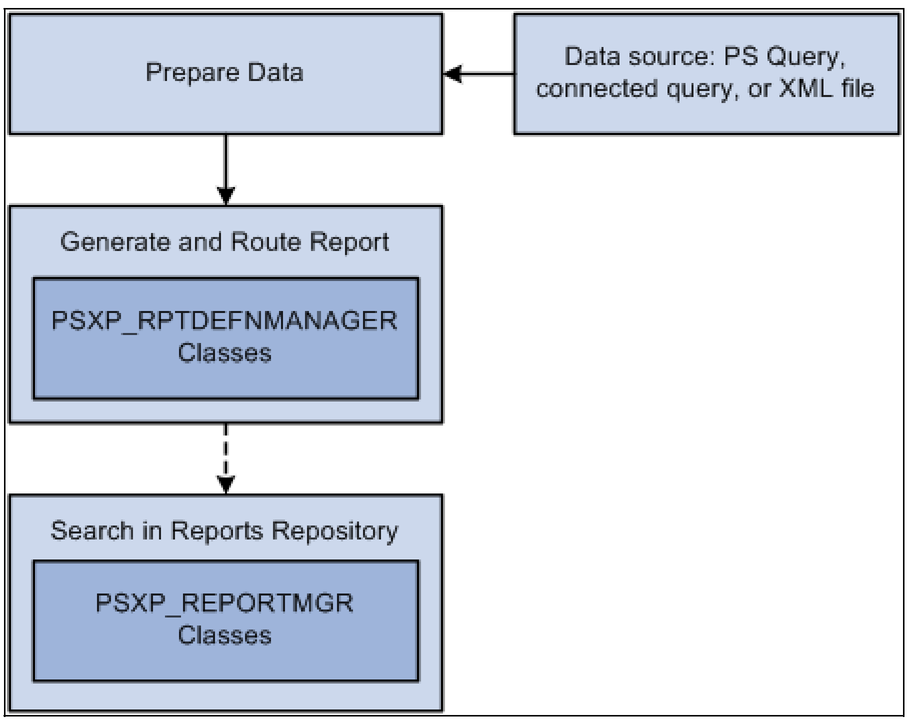
EFT	Electronic Funds Transfer, a format used by the banking industry.
eText	Electronic text, specifically, files in ASCII text format.
FO	Formatting objects, an XML markup language for document formatting.
HTML	Hyper Text Markup Language, a markup language designed for the creation of web pages.
PDF	Portable Document Format, a file format developed by Adobe Systems for representing documents in an independent format.

RTF	Rich Text Format, a document file format used for cross-platform document interchange.
XML	Extendable Markup Language, a general purpose markup language for many types of data.
SMS	Short messaging service, a short alphanumeric (160 characters) that is generally sent to a mobile device, such as a mobile phone.

XML Publisher Flow Diagram

The following is the general flow of an XML report showing how and when certain XML Publisher classes are used in the publishing process.

XML Publisher prepares the report data from the data source: a PS Query, a connected query, or an XML file. Then, XML Publisher uses the PSXP_RPTDEFNMANAGER report definition classes to generate and route the report. Optionally, XML Publisher uses the PSXP_REPORTMGR search classes to search for a report in the reports repository.



XML Publisher flow diagram

Error Handling

Different XML Publisher classes handle errors differently.

Use of report manager definition and report manager search classes should be wrapped in a try-catch statement to handle any errors. The XML Publisher engine class methods typically contain an error parameter as part of their signature that you can check for any errors. Other XML Publisher classes generally return a populated object that can be checked for a null value to determine any errors.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Try

Data Type and Scope of XML Publisher Objects

Each XML Publisher class is its own data type—that is, report definitions are declared as the ReportDefn data type, a report manager is declared as the ReportManager type, and so on.

An XML Publisher object can be declared of type local, component, or global and instantiated from PeopleCode only.

Instantiating XML Publisher Objects

The XML Publisher classes are not built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them into your program.

Your import statements should look similar to the following:

```
import PSXP_RPTDEFNMANAGER:*;
```

Note. Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are not made available.

After you've imported the appropriate XML Publisher classes, you instantiate an object of one of those classes using the constructor for the class and the create method.

The following example declares the variable &rptDefn and creates a new instance of the ReportDefn class:

```
Local PSXP_RPTDEFNMANAGER:ReportDefn &rptDefn = create PSXP_RPTDEFNMANAGER:Report⇒
Defn(&rptDefnId);
```

See Also

Chapter 6, "Application Classes," Constructors, page 204

Chapter 6, "Application Classes," Import Declarations, page 209

Search Operator Values

Methods in the report manager definition and report search manager classes enable you to search for existing definitions. Many of the parameters for these methods should be specified in pairs, that is, a value specified with a search operator. All the search operator parameters use the same values.

The following lists the search operator values that can be used with the XML Publisher classes:

Value	***Description***
%PSXP_SrchBegins	The name to search for begins with the specified value.
%PSXP_SrchContains	The name to search for contains the specified value.
%PSXP_SrchEquals	The name to search for equals the specified value.
%PSXP_SrchNotEquals	The name to search for does not equal the specified value.
%PSXP_SrchLessThan	The name to search for is less than the specified value.
%PSXP_SrchLessEquals	The name to search for is less than or equal to the specified value.
%PSXP_SrchGreaterThan	The name to search for is greater than the specified value.
%PSXP_SrchGreaterEquals	The name to search for is greater than or equal to the specified value.
%PSXP_SrchBetween	The name to search for is between two values. Separate the values by a comma. Do not use quotation marks. For example, `ACCT1,ACCT9`.
%PSXP_SrchIn	The name to search for is in the specified list. Separate the values with commas. Do not use quotation marks. For example, `ACCT1, ACCT2, ACCT3`.

XML Publisher Classes Reference

The XML Publisher classes reference includes the *public* subset of XML Publisher classes divided into the following sections:

- XML Publisher report manager definition classes.
- XML Publisher report manager search classes.
- XML Publisher engine classes.

XML Publisher Report Manager Definition Classes

The following report manager definition classes are described in this chapter: ReportDefn class.

ReportDefn Class

This section provides an overview of the ReportDefn class and discusses:

- ReportDefn class constructor
- ReportDefn class methods
- ReportDefn class properties

Use the ReportDefn class to process a report definition created through the Report Definition component Use this class to generate and publish report output.

ReportDefn Class Constructor

This section describes the constructor for the ReportDefn class.

ReportDefn

Syntax

`ReportDefn`*`(ReportId)`*

Description

Use the ReportDefn constructor to instantiate a ReportDefn object. You must then use the Get method to initialize the object.

Parameters

Parameter	***Description***
ReportId	Specify a unique ID to be associated with the report definition.

Returns

A reference to a ReportDefn object.

Example

```
import PSXP_RPTDEFNMANAGER:ReportDefn;

Local PSXP_RPTDEFNMANAGER:ReportDefn &rptDefn = create PSXP_RPTDEFNMANAGER:Report⇒
Defn(&rptDefnId);
&rptDefn.Get();
```

ReportDefn Class Methods

In this section, the ReportDefn class methods are presented in alphabetical order.

Close

Syntax

Close()

Description

Use this method when there is an exception or error calling the ProcessReport method or any other output method (Publish, PrintOutout or DisplayOutput) of the ReportDefn class. Calling this method cleans up all temporary files, folders, and resources.

Parameters

None.

Returns

None.

See Also

Chapter 47, "XML Publisher Classes," DisplayOutput, page 2602; Chapter 47, "XML Publisher Classes," PrintOutput, page 2604; Chapter 47, "XML Publisher Classes," ProcessReport, page 2605 and Chapter 47, "XML Publisher Classes," Publish, page 2606

DisplayOutput

Syntax

DisplayOutput()

Description

Use the DisplayOutput method to display the report generated by the ProcessReport method in a separate browser window.

You must successfully call the ProcessReport method prior to calling this method.

This method displays a single report. Therefore, the report definition must not be set for bursting.

Parameters

None.

Returns

None.

See Also

Chapter 47, "XML Publisher Classes," Close, page 2601 and Chapter 47, "XML Publisher Classes," ProcessReport, page 2605

Get

Syntax

Get()

Description

Use the Get method to return a reference to the existing ReportDefn object.

Parameters

None.

Returns

A reference to the newly instantiated and populated ReportDefn object.

GetOutDestFormatString

Syntax

`GetOutDestFormatString``(`*`OutDestFormat`*`)`

Description

Use the GetOutDestFormatString method to convert the output format returned from system variable %OutDestFormat to an XML Publisher output format string that can be passed to the output type parameter of the ProcessReport method. This method is useful when processing reports generated by application engine programs that are run process scheduler.

Parameters

Parameter	***Description***
OutDestFormat	Specify the output destination format as a number. If you specify something other than a valid value, the method throws an exception. Valid values are:

Value	***Description***
2	PDF format
5	HTM format
8	XLS format
12	RTF format

Returns

A string if successful. Valid values are:

- PDF
- HTM
- XLS
- RTF

See Also

Chapter 47, "XML Publisher Classes," ProcessReport, page 2605

GetPSQueryPromptRecord

Syntax

GetPSQueryPromptRecord()

Description

Use the GetPSQueryPromptRecord method to return the runtime prompts of a query as a record object. This method should only be used when the data source type associated with the report definition is a PeopleSoft query. This method returns a null when the data source type isn't a PeopleSoft Query.

Parameters

None.

Returns

A record object populated with the query prompts if the data source definition is a PeopleSoft query, null otherwise.

See Also

Chapter 47, "XML Publisher Classes," SetPSQueryPromptRecord, page 2607

PrintOutput

Syntax

PrintOutput*(DestPrinter)*

Description

Use the PrintOutput method to print the report definition object executing the method. The ProcessReport method must be called successfully before you use this method.

Parameters

Parameter	*Description*
DestPrinter	Specify the printer you want to use for printing the report, as a string.

Returns

None.

See Also

Chapter 47, "XML Publisher Classes," Close, page 2601

ProcessReport

Syntax

ProcessReport*(TemplateId,LanguageCD,AsOfDate,OutputFormat)*

Description

Use the ProcessReport method to generate a report and store the information.

Before you generate the report, you must:

- Set the report output destination with the OutDestination property if the output type is a file.
- Specify the data source using the SetRuntimeDataXMLFile method if you are using a data source other than a PeopleSoft query.

After you use the ProcessReport method, you can use the Publish method to post the report, the DisplayOutput method to display the report in a browser window, or the PrintOutput method to print the report.

Parameters

Parameter	*Description*
TemplateId	Specify the template to be used for processing this report, as a string. You can specify a null value for this parameter, that is, (""). When you specify a null value, the default template associated with the report definition is used.

Parameter	Description
LanguageCD	Specify the three-letter language code for the report definition, such as ENG, FRA, ESP and so on. This parameter controls both the translation and the language specific output details such as font selection and right-to-left page orientation for Arabic/Hebrew. You can specify a null value for this parameter, that is, (""). If you specify a null value, the current language of the session is used.
AsOfDate	Specify the as of date for the report, as a date.
OutputFormat	Specify the output format, as a string. See Chapter 47, "XML Publisher Classes," GetOutDestFormatString, page 2603. You can specify a null value for this parameter, that is, (""). When you specify a null value, the default output format associated with the report definition is used.

Returns

None.

See Also

Chapter 47, "XML Publisher Classes," Close, page 2601; Chapter 47, "XML Publisher Classes," DisplayOutput, page 2602; Chapter 47, "XML Publisher Classes," PrintOutput, page 2604; Chapter 47, "XML Publisher Classes," SetPSQueryPromptRecord, page 2607; Chapter 47, "XML Publisher Classes," SetRuntimeDataXMLFile, page 2608 and Chapter 47, "XML Publisher Classes," SetRuntimeProperties, page 2609

Chapter 47, "XML Publisher Classes," OutDestination, page 2611

Publish

Syntax

Publish*(ServerName,ReportPath,FolderName, ProcessInstanceId)*

Description

Use the Publish method to publish the current report definition

You must have successfully completed a call to the ProcessReport method before you can use this method.

Parameters

Parameter	*Description*
ServerName	Specify the server to use for publishing the report, as a string. You can specify a null value for this parameter—that is, ("").
ReportPath	Always specify a null value only for this parameter—that is, ("").
FolderName	Specify the name of the report folder that the report should be posted to. You can specify a null value for this parameter—that is, (""). If you do not provide a value for this parameter, the default folder is used to post reports.
ProcessInstanceId	Specify the process instance of the calling process, such as the calling application engine program. If you specify a 0, a new process instance is generated to track the report posting process.

Returns

None.

See Also

Chapter 47, "XML Publisher Classes," Close, page 2601 and Chapter 47, "XML Publisher Classes," ProcessReport, page 2605

SetPSQueryPromptRecord

Syntax

`SetPSQueryPromptRecord`*`(&Record)`*

Description

Use the SetPSQueryPromptRecord method to specify an already instantiated record object that contains the prompt values for the query to be used to populate the report.

This method can only be used with reports that have a PeopleSoft query defined as the data source.

You must use this method before using the ProcessReport method.

Parameters

Parameter	*Description*
&Record	Specify an already instantiated and populated record object that contains the values for the query to be used to populate the report.

Returns

None.

See Also

Chapter 47, "XML Publisher Classes," ProcessReport, page 2605 and Chapter 47, "XML Publisher Classes," GetPSQueryPromptRecord, page 2604

Chapter 33, "Query Classes," page 1885

SetRuntimeDataXMLFile

Syntax

SetRuntimeDataXMLFile*(FilePath)*

Description

Use the SetRuntimeDataXMLFile method to specify an existing XML file as the data source for the report.

Note. Because the SetRuntimeDataRowset and SetRuntimeDataXMLDoc methods have been deprecated, use this SetRuntimeDataXMLFile method instead.

Report definitions that use a data source other than a PeopleSoft query must set that data source before generating the report using the ProcessReport method.

Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server—that is, on Unix servers, the directory separator is a forward slash, while Windows servers use back slashes.

Parameters

Parameter	*Description*
FilePath	Specify an absolute path to the file that you want to use. You must include the file name and file extension in the parameter. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server use a path with back slashes.

Returns

None.

See Also

Chapter 47, "XML Publisher Classes," ProcessReport, page 2605

Chapter 19, "File Class," page 975

SetRuntimeProperties

Syntax

```
SetRuntimeProperties(&NameArray,&ValueArray)
```

Description

Use the SetRuntimeProperties method to set additional runtime values and properties required for generating this report.

Note. XML Publisher's properties are defined at three different levels. Global properties are defined on the Global Properties page, and system properties are defined in the xdo.cfg file. At the report definition level, properties are defined on the Report Definition - Properties page. The runtime properties defined using this method override both global and report definition properties. However, system properties defined in the xdo.cfg file cannot be overridden using this method.

Parameters

Parameter	*Description*
&NameArray	Specify an already instantiated and populated array of string containing the names of the additional properties or variables that you need for this report.

Parameter	Description
&ValueArray	Specify an already instantiated and populated array of string containing the values of the additional properties or variables that you need for this report. The values should be in the same order as the names listed in *&NameArray*.

Returns

None.

Example

The following example sets extra parameters for securing a document.

```
&asPropName = CreateArrayRept("", 0);
&asPropValue = CreateArrayRept("", 0);
&asPropName.Push("pdf-compression");
&asPropValue.Push("false");
&asPropName.Push("pdf-hide-menubar");
&asPropValue.Push("true");
&oRptDefn.SetRuntimeProperties(&asPropName, &asPropValue);
```

Setting Custom Runtime Parameters

The SetRuntimeProperties method can also be used to set custom runtime parameters. When setting custom runtime parameters with the SetRuntimeProperties method, the parameter names need to be prefixed with *xslt*, and the values need to be surrounded by single quotes (for example, 'xyz'). For example, the following code creates a custom runtime parameter named xslt.ReportOwner:

```
&asPropName = CreateArrayRept("", 0);
&asPropValue = CreateArrayRept("", 0);
&asPropName.Push("xslt.ReportOwner");

/* Note the single quotes around the parameter value */
&asPropValue.Push(" 'John Smith' ");

&oRptDefn.SetRuntimeProperties(&asPropName, &asPropValue);
```

The custom parameter may now be used in the template via the tag <?$ReportOwner?>. Before using it, it does need to be declared in a form field on the report template using the following tag:

```
<xsl:param name="ReportOwner" xdofo:ctx="begin"/>
```

See Also

Chapter 47, "XML Publisher Classes," ProcessReport, page 2605

Enterprise PeopleTools 8.51 PeopleBook: XML Publisher for PeopleSoft Enterprise, "Setting Up XML Publisher," Defining Global Properties; *Enterprise PeopleTools 8.51 PeopleBook: XML Publisher for PeopleSoft Enterprise*, "Defining Report Definitions," Creating Report Definitions; *Enterprise PeopleTools 8.51 PeopleBook: XML Publisher for PeopleSoft Enterprise*, "Setting Up XML Publisher," Defining System Properties and Fonts and *Enterprise PeopleTools 8.51 PeopleBook: XML Publisher for PeopleSoft Enterprise*, "Running, Locating, and Viewing XML Publisher Reports," Passing Parameters

ReportDefn Class Properties

In this section, the ReportDefn class properties are presented in alphabetical order.

Description

Description

Use the Description property to return a description for the report definition.

This property is effectively read-only.

DestinationPrinter

Description

Use the DestinationPrinter property to specify a printer for the generated report.

This property is read-write.

FolderName

Description

Use the FolderName property to specify a folder name for the generated report to be posted to.

This property is read-write.

OutDestination

Description

This property sets the report output destination as a string. This property must be set before calling the ProcessReport method. When ProcessReport is called, it places the generated report files in the directory specified by this property.

If this property is not set prior to calling the ProcessReport method, the report processing directory and the value returned by the OutDestination property depends on the setting of the psxp_usedefaultoutdestination property. The psxp_usedefaultoutdestination property, which is set on the Report Definition - Properties page, is used to set a compatibility mode with pre-8.50 PeopleTools versions.

See *Enterprise PeopleTools 8.51 PeopleBook: XML Publisher for PeopleSoft Enterprise*, "Defining Report Definitions," Setting Report Properties.

This property is effectively write-only.

Important! While this property is read-write, your PeopleCode program should read this property only if the program set the property first. Reading the property before it has been explicitly set produces unexpected results.

See Also

Chapter 47, "XML Publisher Classes," ProcessReport, page 2605

OutDestinationType

Description

This property returns the type of device specified for the output of the report definition, as a string.

This property returns the output destination value set on the Report Definition-Output page; it is not used internally to determine the output type. It can be used in a PeopleCode program to determine how to process and produce report output (for example, whether to call Publish, how to set OutDestination).

This property is effectively read-only.

ProcessInstance

Description

Use this property to specify a number representing a process instance for running the report.

In certain circumstances—for example, when running report using a Process Scheduler server—a process instance number is required for setting drilling URLs in the report output. In these cases, you must set the ProcessInstance property prior to calling the ProcessReport method.

This property is read-write.

Example

```
&ProcessInstance=PSXPQRYRPT_AET.PROCESS_INSTANCE;
&oRptDefn.ProcessInstance = &ProcessInstance;
&oRptDefn.ProcessReport("", "", %Date, "");
```


Status

Description

This property returns the status of the report definition as a string. Valid values are:

Value	*Description*
A	The report definition is active and can be run.
I	The report definition is inactive and cannot be run.
P	The report definition is in progress and cannot be run. This is the default value.

This property is read-only.

UseBurstValueAsOutputFileName

Description

Use the UseBurstValueAsOutputFileName property to specify different names for all the files that result from bursting. This property takes a Boolean value: true, the report output files are named *BurstValue.Extention.* False is the default value.

This property is read-write.

XML Publisher Report Manager Search Classes

The following report manager search classes are described in this chapter:

- Report class
- ReportManager class
- SearchAttribute class

Report Class

This section provides an overview of the Report class and discusses:

- Report class constructor
- Report class properties

The Report class is one of the report manager search classes. Use the Report class to access individual reports that have already been published to the report repository.

Report Class Constructor

This section describes the constructor for the Report class.

Report

Syntax

```
Report(RptId,Prcsinstance,Contentid, Dbname,RptName,RptDescr,RptURL,
RptCreateDttm, RptExpireDt,FldrName)
```

Description

Use the Report constructor to instantiate and populate a Report object with the data of an already published report.

Parameters

Parameter	***Description***
RptId	Specify the report ID of the report you want to access, as a string.
Prcsinstance	Specify the process instance of the report you want to access, as a number.
Contentid	Specify the content ID number of the report you want to access, as a number.
Dbname	Specify the data base name, as a string.
RptName	Specify the name of the report you want to access, as a string.
RptDescr	Specify the description of the report you want to access, as a string.
RptURL	Specify the URL to the report details page, as a string.
RptCreateDttm	Specify the date and time of the report you want to access, as a date time value.
RptExpireDt	Specify the date when the report will be archived, and no longer considered active, as a date.
FldName	Specify the name of the folder the report was published to, as a string. You must specify an absolute path name, using forward and back slashes as required by the operating system of the application server or process scheduler.

Returns

A reference to a Report object.

Example

```
import PSXP_REPORTMGR:Report;

Local PSXP_REPORTMGR:Report &oRpt = create PSXP_REPORTMGR:Report(String(&nRptId),⇒
 &prcsinstance, &contentid, &sdbname, &sRptName, &sRptDescr, &sRptURL, &sFile⇒
URL, &dRptCreateDttm, &dRptExpireDt, &sFldrName);
```

Report Class Properties

In this section, the Report class properties are presented in alphabetical order.

contentId

Description

This property returns the content ID of the report, as a number.

This property is read-only.

CreatedDateTime

Description

This property returns the date time when the report was created, as a date time.

This property is read-only.

DatabaseName

Description

This property returns the data base name associated with the report, as a string.

This property is read-only.

Description

Description

This property returns the description of the report definition, as a string.

This property is read-only.

ExpireDate

Description

This property returns the date when the report will be archived, as a date value.

This property is read-only.

FileURL

Description

This property returns a URL to the actual report file in the report repository.

Use the ReportURL property to access the report detail page instead.

This property is read-only.

See Also

Chapter 47, "XML Publisher Classes," ReportURL, page 2617

FolderName

Description

This property returns the full path name of the folder where the report is stored, as a string.

This property is read-only.

ProcessInstanceID

Description

This property returns the process instance ID associated with the report when it was posted, as a number.

This property is read-only.

See Also

Chapter 47, "XML Publisher Classes," ReportInstanceID, page 2617

ReportInstanceID

Description

This property returns the report instance ID, that was populated when the report was run, as a number.

This property is read-only.

See Also

Chapter 47, "XML Publisher Classes," ProcessInstanceID, page 2617

ReportName

Description

This property returns the name of the report, as a string.

This property is read-only.

ReportURL

Description

This property returns a URL as a string, that points to the report detail page.

Use the FileURL to access a URL to the actual file containing the report in the report repository.

This property is read-only.

See Also

Chapter 47, "XML Publisher Classes," FileURL, page 2616

ReportManager Class

This section provides an overview of the ReportManager class and discusses:

- ReportManager class constructor
- ReportManager class methods

The ReportManager class is one of the report manager search classes. Use the ReportManager class to set the search criteria for reports that have already been published to the report repository.

ReportManager Class Constructor

This section describes the constructor for the ReportManager class.

ReportManager

Syntax

`ReportManager``()`

Description

Use the ReportManager constructor to instantiate an instance of the ReportManager class.

Parameters

None.

Returns

A ReportManager object.

Example

```
import PSXP_REPORTMGR:ReportManager;
```

```
Local REPORTMGR:ReportManager &rptMgr = create REPORTMGR:ReportManager();
```

ReportManager Class Methods

In this section, the ReportManager class methods are presented in alphabetical order.

AddSearchFieldCriteria

Syntax

AddSearchFieldCriteria*(&SearchAttribute)*

Description

Use the AddSearchFieldCriteria method to add an already instantiated and populated SearchAttribute object to the search.

Parameters

Parameter	*Description*
&SearchAttribute	Specify an already instantiated and populated SearchAttribute object.

Returns

A Boolean, true if the criteria was added successfully, false otherwise.

Example

```
Local PSXP_REPORTMGR:SearchAttribute &oSearch;
Local integer &compOpSrch;

&CompOpSrch = oRptMgr.PSXP_SrchBegins;

&oSearch = create PSXP_REPORTMGR:SearchAttribute(&sAttrname, &sAttrVal, ⇒
&compOpSrch);

&bResult = &oRptMgr.addSearchFieldCriteria(&oSearch);
```


See Also

Chapter 47, "XML Publisher Classes," SearchAttribute Class, page 2625

GetReportList

Syntax

GetReportList()

Description

Use the GetReportList method to return a list of the report definition objects that satisfy all the criteria you specified with the other methods associated with the ReportManager object.

Parameters

None.

Returns

An array of ReportDefn objects.

See Also

Chapter 47, "XML Publisher Classes," ReportDefn Class Properties, page 2611

SetBurstFieldCriteria

Syntax

SetBurstFieldCriteria*(BurstFld,BurstOp,BurstValue)*

Description

Use the SetBurstFieldCriteria method to specify the burst criteria to be used with searching report definitions.

Parameters

Parameter	*Description*
BurstField	Specify the name of the burst field used to generate the report, as a string.
BusrtOp	Specify the search operator to be used *BurstField,* as a string. See Chapter 47, "XML Publisher Classes," Search Operator Values, page 2599.
BurstValue	Specify the pattern string to be matched with burst field value using *BurstOp.*

Returns

A Boolean value, true if the method completes successfully, false otherwise.

SetCaseSensitive

Syntax

`SetCaseSensitive``(`*`IsCaseSensitive`*`)`

Description

Use the SetCaseSensitive method to specify the case sensitivity flag applicable to search field and bursting criteria for searching reports.

Parameters

Parameter	*Description*
IsCaseSensitive	Specify a Boolean value: true to make the search case sensitive, false to make it case insensitive.

Returns

None.

SetDateCriteria

Syntax

```
SetDateCriteria(createdDate,createdLastVal,createdUnit)
```

Description

Use the SetDateCriteria method to specify the creation date criteria for searching reports.

Parameters

Parameter	***Description***
createdDate	Specify a date used to search for reports generated on and after this date. You should specify a null value (" ") for this parameter if using *createdLastVal* and *createdUnit.*
createdLastVal	Specify a value to be used n conjunction with *createdUnit*, as a number. If you specify a value, it is used and any value given for *createdDate*is ignored. If you are not using this search criteria, you should specify a null value (" ") for this parameter.
createdUnit	Specify a value to be used for reports generated during last *n* number of periodic units. This value is used with *createdLasVal.* For example, if *createdLastVal* is set to 10, and *createdUnit* is set to 1, the reports generated in the last ten days are returned. Valid values are:

Value	***Description***
1	Days
2	Hours
3	Minutes

Returns

None.

SetFolderCriteria

Syntax

SetFolderCriteria*(FolderName)*

Description

Use the SetFolderCriteria method to specify the folder criteria for searching reports.

Parameters

Parameter	***Description***
FolderName	Specify the name of the folder the report is posted in, as a string.

Returns

A Boolean value, true if the search criteria is added successfully, false otherwise.

SetProcessInstanceCriteria

Syntax

SetProcessInstanceCriteria*(FromPID,ToPID)*

Description

Use the SetProcessInstanceCriteria method to specify the process instance range criteria for searching reports.

Parameters

Parameter	***Description***
FromPID	Specify a process instance ID to be used as the starting range from which to search, as a number. This value must be less than or equal to *ToPID.* You can specify a null value for this parameter, that is, (" ").
ToPID	Specify a process instance ID to be used as the ending range from which to search, as a number. This value must be greater than or equal to *FromPID.* You can specify a null value for this parameter, that is, (" ").

Returns

A Boolean value: true if the search criteria is added successfully, false otherwise.

SetReportIDCriteria

Syntax

`SetReportIDCriteria`*`(ReportId)`*

Description

Use the SetReportIDCriteria method to specify a report ID for searching reports.

Parameters

Parameter	***Description***
ReportId	Specify a report ID to be used for searching for report definitions.

Returns

A Boolean value: true if the search criteria was set successfully, false otherwise.

SetUserIdCriteria

Syntax

`SetUserIdCriteria`*`(UserId)`*

Description

Use the SetUserIdCriteria method to specify a user ID to be used for searching reports intended to be accessed by that recipient.

Parameters

Parameter	***Description***
UserId	Specify the user ID to be matched with the intended recipient of the report.

Returns

A Boolean: true if the search criteria is set successfully, false otherwise.

SearchAttribute Class

This section provides an overview of the SearchAttribute class and discusses:

- SearchAttribute class constructor
- SearchAttribute class properties

The SearchAttribute class is one of the report manager search classes. Use the SearchAttribute class to discover name-value pairs for the search criteria for reports that have already been published to the report repository. In addition, you can also specify a comparison operator to be used with the name-value pairs.

SearchAttribute Class Constructor

This section describes the constructor for the SearchAttribute class.

SearchAttribute

Syntax

`SearchAttribute`*`(AttrName,AttrValue,CompareOperator)`*

Description

Use the SearchAttribute constructor to instantiate an instance of a SearchAttribute object.

Parameters

Parameter	***Description***
AttrName	Specify the attribute name, as a string.
AttrValue	Specify the attribute value, as a string.
CompareOperator	Specify a comparison operator. See Chapter 47, "XML Publisher Classes," Search Operator Values, page 2599.

Returns

A SearchAttribute object.

Example

```
import PSXP_REPORTMGR:SearchAttribute;

Local PSXP_REPORTMGR:SearchAttribute &oSearch = create PSXP_REPORTMGR:Search⇒
Attribute(&sAttrname, &sAttrVal, &compOpSrch);
```

SearchAttribute Class Properties

In this section, the SearchAttribute class properties are presented in alphabetical order.

attrName

Description

This property returns the attribute name used with search.

This property is read-only.

attrValue

Description

This property returns the attribute value used with search???

This property is read-only.

compareOp

Description

This property returns the comparison operator used with search.

This property is read-only.

XML Publisher Engine Classes

The following engine classes are described in this chapter:

- PageNumber class
- PDFMerger class
- Properties class
- Watermark class

PageNumber Class

This section provides an overview of the PageNumber class and discusses:

- PageNumber class constructor
- PageNumber class properties

Use the PageNumber class to set the page number for a merged PDF file. Objects in this class are used with the PageNumber property of the PDFMerger class.

See Also

Chapter 47, "XML Publisher Classes," PageNumber, page 2634

PageNumber Class Constructor

This section describes the constructor for the PageNumber class.

PageNumber

Syntax

PageNumber()

Description

Use the PageNumber constructor to instantiate a PageNumber object.

Parameters

None.

Returns

A PageNumber object.

Example

```
import PSXP_ENGINE:PageNumber;

Local PSXP_ENGINE:PageNumber &pNum = create PSXP_ENGINE:PageNumber();
```

PageNumber Class Properties

In this section, the PageNumber class properties are presented in alphabetical order.

BackgroundFile

Description

Use the BackgroundFile property to specify a file to be used for the page number, as a string.

You must specify an absolute file name.

This property is read-write.

Considerations Using the BackgroundFile Property

You should either use the BackgroundFile property, or you should use the other properties associated with this class, like position or font. You cannot use both.

The file you specify must be a PDF file. The page size of the background PDF must be the same as the target PDF document, otherwise the page numbering doesn't work properly. All page numbering starts on the first page of the target document.

FontName

Description

Use the FontName property to specify the name of the font to be used for the page number, as a string. Valid values are:

- Courier
- Courier-Bold
- Courier-BoldOblique
- Helvetica (this is the default value)
- Helvetica-Bold
- Helvetica-BoldOblique
- Helvetica-Oblique
- Symbol
- Times-Bold
- Time-BoldItalic
- Time-Italic
- Time-Roman
- ZapfDingbats

If you specify an invalid font name, the default (Helvetica) is used.

This property is read-write.

See Also

Chapter 47, "XML Publisher Classes," FontSize, page 2629

FontSize

Description

Use the FontSize property to specify the size of the page number, as a number.

The default value is 8.

This property is read-write.

See Also

Chapter 47, "XML Publisher Classes," FontName, page 2628

PositionX

Description

Use the PositionX property to specify the X axis position of the text page number in the merged document.

This property is read-write.

See Also

Chapter 47, "XML Publisher Classes," PositionY, page 2630

PositionY

Description

Use the PositionY property to specify the Yaxis position of the text page number in the merged document.

This property is read-write.

See Also

Chapter 47, "XML Publisher Classes," PositionX, page 2630

StartFromPageNum

Description

Use the StartFromPageNum property to specify the page index from which you'd like to start the page numbering.

For example, if you have a PDF document which has two cover pages, and you want to start printing page numbers on the document from the third page, specify a three for this property.

This property is read-write.

See Also

Chapter 47, "XML Publisher Classes," StartNum, page 2631

StartNum

Description

Use the StartNum property to specify the page number to use as the first page number in the merged document. If you don't specify a starting number, the pages are numbered starting from 1.

This property is read-write.

See Also

Chapter 47, "XML Publisher Classes," StartFromPageNum, page 2630

PDFMerger Class

This section provides an overview of the PDFMerger class and discusses:

- PDFMerger class constructor
- PDFMerger class method
- PDFMerger class properties

Use the PDFMerger class to combine two or more PDF files into a single PDF file.

PDFMerger Class Constructor

This section describes the constructor for the PDFMerger class.

PDFMerger

Syntax

PDFMerger()

Description

Use the PDFMerger constructor to instantiate a PDFMerger object.

Parameters

None.

Returns

A reference to a PDFMerger object.

Example

```
import PSXP_ENGINE:PDFMerger;

Local PSXP_ENGINE:PDFMerger &oMerger = create PSXP_ENGINE:PDFMerger();
```

PDFMerger Class Method

In this section, the PDFMerger class method is presented.

MergePDFs

Syntax

MergePDFs*(PDFFileArray,PDFOutputFile,Error)*

Description

Use the MergePDFs method to merge the specified PDF files into a single output file.

The order of the files specified in the array are the order in which the files are merged.

Parameters

Parameter	***Description***
PDFFileArray	Specify an already instantiated and populated array of string containing the names of the PDF files that you want to merge together.
PDFOutputFile	Specify the full path name of the file you want populated with the merged PDF file. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.

Parameter	*Description*
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A Boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

Example

```
Local PSXP_ENGINE:PDFMerger &oMerger;

&sErr = "";
&asNames = CreateArray(&sPdfFile1);
&asNames.Push(&sPdfFile2);
&bResult = &oMerger.mergePDFs(&asNames, &sOutputPdfFile, &sErr);
```

PDFMerger Class Properties

In this section, the PDFMerger class properties are presented in alphabetical order.

ConfProp

Description

Use the ConfProp property to specify an already instantiated and populated Properties object that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

See Also

Chapter 47, "XML Publisher Classes," Properties Class, page 2635

Locale

Description

Use the Locale property to specify the locale for the processing. If no value is specified for this property, the default system locale will be used.

The locale is specified using a two character ISO language code and a two character ISO country code (LC-CC). When you don't need to reflect country specific formatting, the locale code is made up of just the language code.

This property is read-write.

PageNumber

Description

Use this property to specify an already instantiated and populated PageNumber object to be used with the merged document.

This property is read-write.

Example

```
Local PSXP_ENGINE:PDFMerger &oMerger;
Local PSXP_ENGINE:PageNumber &oPageNumber;

&oMerger = create PSXP_ENGINE:PDFMerger();
&oPageNumber = create PSXP_ENGINE:PageNumber();

&oPageNumber.FontName = "Symbol";
&oPageNumber.FontSize = 16;
&oPageNumber.PositionX = 300;
&oPageNumber.PositionY = 20;
&oMerger.oPageNumber = &oPageNumber;

&sErr = "";
&asNames = CreateArray(&sPdfFile1);
&asNames.Push(&sPdfFile2);
&bResult = &oMerger.mergePDFs(&asNames, &sOutputPdfFile, &sErr);
```

See Also

Chapter 47, "XML Publisher Classes," PageNumber Class, page 2627

Watermark

Description

Use this property to specify an already instantiated and populated Watermark object to be used with the merged object.

This property is read-write.

Example

```
Local PSXP_ENGINE:PDFMerger &oMerger;
Local PSXP_ENGINE:Watermark &oWatermark;

&oMerger = create PSXP_ENGINE:PDFMerger();
&oWatermark = create PSXP_ENGINE:Watermark();

&oWatermark.Text = "DEMO";
&oWatermark.TextStartPosX = 200;
&oWatermark.TextStartPosY = 400;
&oMerger.oWatermark = &oWatermark;

&sErr = "";
&asNames = CreateArray(&sPdfFile1);
&bResult = &oMerger.mergePDFs(&asNames, &sOutputPdfFile, &sErr);
```

See Also

Chapter 47, "XML Publisher Classes," Watermark Class, page 2638

Properties Class

This section provides an overview of the Properties class and discusses:

- Properties class constructor
- Properties class methods

Use the Properties class to specify the properties for setting up the processor configuration. This class is used with the PDFMerger class.

See Also

Chapter 47, "XML Publisher Classes," ConfProp, page 2633

Properties Class Constructor

This section describes the constructor for the Properties class.

Properties

Syntax

Properties()

Description

Use the Properties constructor to instantiate a Properties object.

Parameters

None.

Returns

A Properties object.

Example

```
import PSXP_ENGINE:Properties;

Local PSXP_ENGINE:Properties &rProps = create PSXP_ENGINE:Properties();
```

Properties Class Methods

In this section, the Properties class methods are presented in alphabetical order.

GetProperty

Syntax

`GetProperty`*`(Name,OutValue)`*

Description

Use the GetProperty method to return the value of the specified property.

Parameters

Parameter	***Description***
Name	Specify the name of the property that you want to access the value.
OutValue	Specify a string to contain the value of the property you want to access.

Returns

A Boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

Example

```
Local PSXP_ENGINE:Properties &oProp;

&oProp = create PSXP_ENGINE:Properties();
%sValue = "";
&bResult = &oProp.getProperty("pdf-security", %sValue);
```

SetProperty

Syntax

`SetProperty`*`(Name,Value)`*

Description

Use the SetProperty method to specify the value of the specified property.

Parameters

Parameter	***Description***
Name	Specify the name of the property that you want to set.
Value	Specify the value of the property that you want to set.

Returns

A Boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

Example

```
Local PSXP_ENGINE:Properties &oProp;

&oProp = create PSXP_ENGINE:Properties();
&bResult = &oProp.setProperty("pdf-security", "true");
```


Watermark Class

This section provides an overview of the Watermark class and discusses:

- Watermark class constructor
- Watermark class properties

Use the Watermark class to specify a watermark for a merged PDF file. Objects in this class are used with the Watermark property of the PDFMerger class. You can generate a watermark either using text or an image. You can either use the text properties or the image properties. You cannot use both.

The following are the image properties of the Watermark class:

- ImageFile
- ImageLowerLeftX
- ImageLowerLeftY
- ImageUpperRightX
- ImageUpperRightY

The following are the text properties of the Watermark class:

- Text
- TextAngle
- TextFontName
- TextFontSize
- TextStartPosX
- TextStartPosY

See Also

Chapter 47, "XML Publisher Classes," Watermark, page 2634

Watermark Class Constructor

This section describes the constructor for the Watermark class.

Watermark

Syntax

`Watermark``()`

Description

Use the Watermark constructor to instantiate a Watermark object.

Parameters

None.

Returns

A Watermark object.

Example

```
import PSXP_ENGINE:Watermark;

Local PSXP_ENGINE:Watermark &rProps = create PSXP_ENGINE:Watermark();
```

Watermark Class Properties

In this section, the Watermark class properties are presented in alphabetical order.

ImageFile

Description

Use the ImageFile property to specify the full path name of an image file to be used for the watermark.

Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.

This property is read-write.

ImageFileLowerLeftX

Description

Use the ImageFileLowerLeftX property to specify the lower left, X-axis position for the watermark, as a number.

This property is read-write.

ImageFileLowerLeftY

Description

Use the ImageFileLowerLeftY property to specify the lower left, Y-axis position for the watermark, as a number.

This property is read-write.

ImageFileUpperRightX

Description

Use the ImageFileUpperRightX property to specify the upper right, X-axis position for the watermark, as a number.

This property is read-write.

ImageFileUpperRightY

Description

Use the ImageFileUpperRightY property to specify the upper right, Y-axis position for the watermark, as a number.

This property is read-write.

PageIndex

Description

Use the PageIndex property to specify the page where the watermark begins. For example, you might not want a watermark to display on a cover page. The default value is 0, which sets the watermark on all pages (page index starts at 1.)

This property is read-write.

Text

Description

Use the Text property to specify the text for the watermark, as a string.

This property is read-write.

TextAngle

Description

Use the TextAngle property to specify the angle of the watermark text across the page. You must specify a number between 1 and 360.

This property is read-write.

TextFontName

Description

Use the TextFontName property to specify the name of the font to be used for the text for the watermark.

This property is read-write.

TextFontSize

Description

Use the TextFontSize property to specify the size of the text font. The default value is 8.

This property is read-write.

TextStartPosX

Description

Use the TextStartPosX property to specify the starting position, X-axis location for the text to be used as the watermark.

This property is read-write.

TextStartPosY

Description

Use the TextStartPosY property to specify the starting position, Y-axis location for the text to be used as the watermark.

This property is read-write.

XML Publisher Classes Example

The following example provides a complete example of the code first, then goes through the program line by line.

Generating and Publishing a Report

In the following example, a report is created from an existing report definition. It is populated with an already instantiated and populated XmlDoc object, then published.

The following is the complete code sample: the steps explain each line.

```
import PSXP_RPTDEFNMANAGER:ReportDefn;

Local string &sFileName = "c:\path\filename.xml";
Local string &rptDefnId = "Financial";
Local string &LanguageCode = "ENG";
Local date &effdt = Date(20090821);
Local string &outputfmt = "HTM";

Local string &folderName = "General";
Local string &serverName = "PSNT";
Local PSXP_RPTDEFNMANAGER:ReportDefn &rptDefn;

&rptDefn = create PSXP_RPTDEFNMANAGER:ReportDefn(&rptDefnId);
&rptDefn.Get();
&rptDefn.SetRuntimeDataXMLFile(&sFileName);
&rptDefn.ProcessReport("", &languageCd, &effdt, &outputfmt);

&rptDefn.Publish(&serverName, "", &folderName, &PID);
```


To generate and publish a report:

1. Import the appropriate application class.

   Because this program generates and publishes a report, you need to import the report manager definition class.

   ```
   import PSXP_RPTDEFNMANAGER:ReportDefn;
   ```

2. Initialize variables.

   The variable declaration strings not only specify values for the variables, but give them type and scope as well. This can be very useful when debugging.

   ```
   Local string &sFileName = "c:\path\filename.xml";
   Local string &rptDefnId = "Financial";
   Local string &LanguageCode = "ENG";
   Local date &effdt = Date(20090821);
   Local string &outputfmt = "HTM";

   Local string &folderName = "General";
   Local string &serverName = "PSNT";
   Local PSXP_RPTDEFNMANAGER:ReportDefn &rptDefn;
   ```

3. Instantiate the report definition object and initialize it.

   After you instantiate a report definition object, you must initialize it and populate it using the Get method.

   ```
   &rptDefn = create PSXP_RPTDEFNMANAGER:ReportDefn(&rptDefnId);
   &rptDefn.Get();
   ```

4. Specify the data for the report.

   This report uses an XML file as the data source, so you must specify the runtime data source for the report before you process it.

   ```
   &rptDefn.SetRuntimeDataXMLFile(&sFileName);
   ```

5. Process the report.

   You must process the report, generate a version of it for the report repository, before you can distribute the report.

   ```
   &rptDefn.ProcessReport("", &languageCd, &effdt, &outputfmt);
   ```

6. Publish the report.

   After you've generated the report, you may want to publish it to another location.

   ```
   &rptDefn.Publish(&serverName, "", &folderName, &PID);
   ```

Appendix A

Quick Reference for PeopleCode Classes

This appendix discusses:

- PeopleCode syntax quick reference
- PeopleCode classes alphabetical reference.
- Session classes.
- Deprecated items and PeopleCode no longer supported.

PeopleCode Syntax Quick Reference

The following is a description of the PeopleCode syntax, as derived from the PeopleCode parser. This description uses these metacharacters:

Metacharacter	***Description***
[ ]	Brackets are used to indicate optional portions.
...	Ellipses are used to indicate that the preceding part (often optional) repeats an arbitrary number of times.
|	Vertical bars are used to separate alternatives. They indicate that one of the alternatives should be present. The alternatives extend only to the immediately enclosing parentheses or brackets (if any).
()	Parentheses are used to group portions of definitions to limit the scope of alternatives (|) or optionality (...).
'symbols'	Quoted characters in **keyword style** are keywords or punctuation in the language. They should appear literally, without the quotation marks.

The following is the PeopleCode language syntax quick reference:

Term	***Syntax***
Program	ImportList (DeclList TopStList | ClassDefn)
ImportList	[ImportDecl **';'**]...

Term	***Syntax***
ImportDecl	**'import'** PackageName [**':'** **'*'**]
PackageName	Id [**':'** Id]...
ClassDefn	ClassDecl [**';'** ExtDecls] [**';'** MethodDefns]
ClassDecl	**'class'**| **'interface''** Id [Extends] [ClassPublics] [ClassProtecteds] [ClassPrivates] **'end-class'** | **'end-interface'**
Extends	**'extends'** | **'implements'** QualifiedId
QualifiedId	[PackageName **':'**] Id
ClassPublics	ClassPublic [**';'** ClassPublic]...
ClassPublic	MethodDecl | PropertyDecl
MethodDecl	**'method'** Id **'('** [MethodParameters] **')'** [**'abstract'**] [**'returns'** PeopleCodeType]
MethodParameters	MethodParameter [**','** MethodParameter]...
MethodParameter	**'&'** Id **'as'** PeopleCodeType [**'out'**]
PropertyDecl	**'property'** PeopleCodeType Id [**('get'** [**'set'**] | **'abstract')** | **'readonly'**]
ClassPrivates	**'private'** [ClassPrivate [**';'** ClassPrivate]...]
ClassPrivate	MethodDecl | InstanceDecl | ConstantDecl
ClassProtecteds	**'protected'** [ClassProtected [**';'** ClassProtected]...]
ClassProtected	MethodDecl | InstanceDecl | ConstantDecl
InstanceDecl	**'instance'** VarDeclare
ExtDecls	ExtDecl [**';'** ExtDecl]...
MethodDefns	MethodDefn [**';'** MethodDefn]...
MethodDefn	MethodMethod | GetMethod | SetMethod
MethodMethod	**'method'** Id StList **'end-method'**
GetMethod	**'get'** Id StList **'end-get'**
SetMethod	**'set'** Id StList **'end-set'**

Term	***Syntax***
TopStList	TopStmt [**';'** TopStmt]...
TopStmt	Stmt
StList	Stmt [**';'** Stmt]...
Stme	[LValue [Assign] | If | Evaluate | While | Repeat | For | Accept | Break | Return | Exit | Error | Warning | Try | Throw | LocalDecl]
Assign	**'='** Expression
If	**'if'** LogicalExpression **'then'** StList [**'else'** StList] **'end-if'**
Evaluate	**'evaluate'** Expression [WhenExpr... StList]... [**'when-other'** StList] **'end-evaluate'**
WhenExpr	**'when'** [RelOp] Expression
While	**'while'** LogicalExpression StList **'end-while'**
Repeat	**'repeat'** StList **'until'** LogicalExpression
For	**'for'** Variable **'='** Expression **'to'** Expression [**'step'** Expression] StList **'end-for'**
Accept	**'accept'**
Return	**'return'** [Expression]
Break	**'break'**
Exit	**'exit'** [Expression]
Warning	**'warning'** Expression
Error	**'error'** Expression
Try	**'try'** StList **('catch'** QualifiedId **'&'** Id StList)... **'end-try'**
Throw	**'throw'** Expresssion
LocalDecl	**'local'** PeopleCodeType **'&'** Id [**(','** **'&'** Id)... | '=' Expression]
Continue	**'Continue'**
LogicalExpression	LogicalTerm [**'or'** LogicalTerm]...

<table>
<tr><th>Term</th><th>Syntax</th></tr>
<tr><td>LogicalTerm</td><td>Relation ['and' Relation]...</td></tr>
<tr><td>Relation</td><td>'not' Relation | Expression [Relop Expression]</td></tr>
<tr><td>Relop</td><td>'not' Relop | '<' | '<=' | '=' | '>=' | '>' | '!=' | '<>'</td></tr>
<tr><td>Expression</td><td>Term [('+' | '-' | '|') Term]...</td></tr>
<tr><td>Term</td><td>Power [('*' | '/') Power]...</td></tr>
<tr><td>Power</td><td>Primary ['**' Primary]...</td></tr>
<tr><td>Primary</td><td>LValue ['as' QualifiedId] | Number | String | 'true' | 'false' | 'null' | '-' Primary | 'create' QualifiedId Call</td></tr>
<tr><td>LValue</td><td>LValueStart [LValueObject | LValueSubscript]...</td></tr>
<tr><td>LValueStart</td><td>Variable | Id Call | '@' Primary | '%' BuiltinVar | '(' LogicalExpression ')'</td></tr>
<tr><td>LValueObject</td><td>'.' Id [Call] | Call</td></tr>
<tr><td>LValueSubscript</td><td>'[' Expression [',' Expression]... ']'</td></tr>
<tr><td>Variable</td><td>'&' Id | Id ['.' (Id | String)] | '^'</td></tr>
<tr><td>Call</td><td>'(' [Expression [',' Expression]...] ')'</td></tr>
<tr><td>Constant</td><td>Number | String | 'true' | 'false' | 'null' | QualifiedId ':' Id</td></tr>
<tr><td>DecList</td><td>[Decl ';']...</td></tr>
<tr><td>Decl</td><td>'local' VarDeclare | 'function' Function | ConstDecl | ExtDecl</td></tr>
<tr><td>VarDeclare</td><td>PeopleCodeType '&' Id [',' '&' Id]...</td></tr>
<tr><td>Function</td><td>Id ['(' InternalParameters ')'] ['returns' PeopleCodeType] ['noexport'] ['doc' String] TopStList 'end-function'</td></tr>
<tr><td>InternalParameters</td><td>[InternalParameter [',' InternalParameter]...]</td></tr>
<tr><td>InternalParameter</td><td>'&' Id ['as' PeopleCodeType]</td></tr>
<tr><td>ConstDecl</td><td>'constant''&' Id '=' (Number | String | 'true' | 'false' | 'null')</td></tr>
</table>

Term	***Syntax***
ExtDecl	**'global'** VarDeclare | **'component'** VarDeclare | **'declare'** Declare
Declare	**'function'** Id **('library'** DeclareLibrary | **'peoplecode'** DeclarePC)
DeclareLibrary	String [**'alias'** String] [**'('** ExternalParameters **')'**] [**'returns'** ExternalType [**'as'** PeopleCodeType]]
ExternalParameters	[ExternalParameter [**','** ExternalParameter]...]
ExternalParameter	ExternalType [**'value'** | **'ref'**] [**'as'** PeopleCodeType]
DeclarePC	Variable EventType
PeopleCode Type	[**'array'** **'of'**]... **('number'** | **'string'** | **'date'** | **'time'** | **'datetime'** | **'any'** | **'boolean'** | **'object'** | **'array'** | **'integer'** | **'float'** | QualifiedId)
ExternalType	**'boolean'** | **'integer'** | **'long'** | **'uinteger'** | **'ulong'** | **'string'** | **'lstring'** | **'ustring'** | **'float'** | **'double'**

PeopleCode Classes Alphabetical Reference

This section provides an overview of typographical conventions and visual cues and lists the PeopleCode classes in alphabetical order, along with the functions, methods and properties associated with that class.

Typographical Conventions and Visual Cues

This table describes the typographical conventions and visual cues that are used in this quick reference:

Typographical Convention or Visual Cue	***Description***
D	Identifies the default method for a class.
RO	Identifies a read-only property. Properties that aren't identified as read-only are read-write.

In addition, the usual PeopleCode typographical conventions are used to distinguish between different elements of the PeopleCode language, such as **keyword** to indicate keywords or syntax that must be entered exactly as shown, *variable* for arguments, parameters, and so on.

See Also

PeopleTools 8.51 PeopleBook: PeopleCode Language Reference, "PeopleCode Language Reference Preface," PeopleCode Typographical Conventions

AESection

This section lists the functions, methods, and properties, as well as returns (if applicable) for the AESection class.

Function

Function	***Returns***
GetAESection*(applid,ae_section* [, *effdt*])	AESection object

Methods

Method	***Returns***
AddStep*(ae_step_name* [,*NewStepName*])	None
Close()	None
Open(*ae_applid,ae_section,* [*effdt*])	AESection object
Save()	None
SetSQL*(action_type_string,string)*	None
SetTemplate*(ae_applid,ae_section)*	None

Property

Property	***Returns***
IsOpen	Boolean [RO]

Analytic Calculation Engine Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the Analytic Calculation Engine Classes.

Important! The Analytic Calculation Engine classes are not supported on IBM z/OS and Linux for IBM System z.

Functions

Function	***Returns***
CreateAnalyticInstance*(AnalyticType,AIID,Descr, AppClassName, MethodName,&RecordRef,Force)*	AnalyticInstance object
GetAnalyticInstance*(AIID)*	AnalyticInstance object

AnalyticInstance Methods

Method	***Returns***
CheckAsyncStatus()	Number
CheckStatus()	Constant
Copy*(NewAIID,ForceDelete,&Record)*	None
Delete*(ForceUnload,&RecordRef)*	None
GetAnalyticModel(**(Model.***ModelName)*	AnalyticModel object
GetTraceLevel()	Integer
Load((**Message.***MessageName*), *Sync*,*IdleTimeOut*)	String
RunAsync()	None
RunSync()	None
SetTraceLevel*(TraceLevel)*	Number
Terminate()	None.
Unload()	None

AnalyticInstance Properties

Property	***Returns***
AnalyticType	String^RO
ID	Integer^RO
Messages	Multi-dimensional array of any^RO

AnalyticModel Methods

Method	***Returns***
AddMember*(DimName,MemberName)*	None
AttachTree*(DimName,TreeName,SetId,UserKeyValue, EffDt,NodeName,OverrideRecord,DetailStartLvl, TreeDiscardLvl*)	None
CalculateCube*(CubeName,Sync)*	None
DetachTree*(DimName)*	None
GetCubeCollection*(CubeCollName)*	CubeCollection object
GetCellProperties*(CubeName,&Node)*	Array of string
GetMembers*(DimName,DimFilter)*	Two-dimensional array
GetTree*(DimName)*	Array of string
Recalculate*(Sync)*	String
RenameMember*(DimName,OrigMemberName, NewMemberName)*	None

AnalyticModel Property

Property	***Returns***
Messages	Multi-dimensional array of any^RO

CubeCollection Methods

Method	***Returns***
CollapseNode*(DimName,&Node)*	None
DrillIntoNode*(DimName,&Node)*	None

Method	***Returns***
DrillOutOfNode*(DimName,&Node)*	None
ExpandNode*(DimName,&Node,ExpandAll)*	None
GetData*(&DataRowset,StartRow,EndRow*[, *NetChanges*])	None
GetDimFilter*(DimName)*	String
GetDimSort*(DimName)*	Array
GetLayout*(&SlicerArray,&RowAxisArray, &ColumnAxisArray)*	None
GetRowCount()	Integer
GetSlice*(&SliceRecord)*	None
SetData*(&DataRowset)*	None
SetDimensionOrder*(&FieldNames)*	None
SetDimFilter*(DimName,DimFilter)*	None
SetDimSort(*DimName,IsAscending*[, *Key1,IsAscending2, Key2,IsAscending3,Key3*])	None
SetLayout*(&SlicerArray,&RowAxisArray, &ColumnAxisArray*	None
SetSlice(*&SliceRecord)*	None
ShowHierarchy*(DimName,Show)*	None
UnsetDimFilter*(DimName)*	None
UnsetDimSort*(DimName)*	None

CubeCollection Property

Property	***Returns***
Messages	Multi-dimensional array of any^{RO}

Analytic Calculation Engine Metadata Classes

This section lists the methods and properties, as well as returns (if applicable) for the Analytic Calculation Engine metadata classes.

Important! The Analytic Calculation Engine Metadata classes are not supported on IBM z/OS and Linux for IBM System z.

AnalyticModelDefn Methods

Method	***Returns***
AddCube*(CubeName)*	CubeDefn object
AddCubeCollection*(CubeCollectionName)*	CubeCollectionDefn object
AddDimension*(DimName)*	DimensionDefn object
AddExplicitDimensionSet *(ExplicitDimSetName)*	ExplicitDimensionSet object
AddOrganizer*(OrganizerName)*	OrganizerDefn object
AddUserFunction*(UserFunctionName)*	UserFunction object
CopyCube*(CubeName,NewCubeName)*	CubeDefn object
CopyCubeCollection*(CubeCollectionName, NewCubeCollectionName)*	CubeCollectionDefn object
CopyDimension*(DimName,NewDimName)*	DimensionDefn object
CopyExplicitDimensionSet*(OldExplicitDimSetName, NewExplicitDimSetName)*	ExplicitDimensionSet object
CopyTo*(NewModelName)*	AnalyticModelDefn object
CopyUserFunction*(UserFunctionName, NewUserFunctionName)*	UserFunctionDefn object
Create()	None
Delete()	None
DeleteCube*(CubeName,ForceDelete)*	None
DeleteCubeCollection*(CubeCollectionName,ForceDelete)*	None
DeleteOrganizer*(OrganizerName)*	None
DeleteDimension*(DimensionName)*	None
DeleteExplicitDimensionSet*(ExplicitDimensionSetName)*	None
DeleteUserFunction*(UserFunctionName,ForceDelete)*	None

Method	***Returns***
Get()	None
GetCube*(CubeName)*	CubeDefn object
GetCubeCollection*(CubeCollection)*	CubeCollectionDefn object
GetCubeCollectionNames()	Array of string
GetCubeNames()	Array of string
GetDimension*(DimName)*	DimensionDefn object
GetDimenstionNames()	Array of string
GetExplicitDimensionSet	ExplicitDimensionSet object
GetExplicitDimensionSetNames	Array of strings
GetOrganizer*(OrganizerName)*	OrganizerDefn object
GetOrganizerNames()	Array of string
GetUserFunction*(UserFunctionNames)*	UserFunctionDefn object
GetUserFunctionNames()	Array of string
Rename*(NewModelName)*	None
RenameCube*(CubeName,NewCubeName,ForceRename)*	None
RenameCubeCollection*(CubeCollectionName, NewCubeCollectionName,ForceRename)*	None
RenameDimension*(DimensionName,NewDimName, ForceRename)*	None
RenameExplicitDimensionSet*(ExplicitDimSetName, NewExplicitDimSetName,ForceRename*)	None
RenameOrganizer*(OrganizerName,NewOrganizerName, ForceRename)*	None
RenameUserFunction*(UserFunctionName, NewUserFunctionName,ForceRename)*	None
Save()	None
Validate()	Boolean

AnalyticModelDefn Properties

Property	***Returns***
CircularFomulaWarn	Boolean
Description	String
IsValid	Boolean^{RO}
LongDescription	String
MaxDelta	Integer
MaxIterations	Integer
Messages	Multi-dimensional array of any^{RO}
Name	String^{RO}
ResolveCircularDeps	Boolean

DimensionDefn Properties

Property	***Returns***
AggregateSequence	Number^{RO}
AggregationUserFunction	String
Comments	String
Name	String^{RO}
TotalMemberName	String

ExplicitDimensionSet Methods

Method	***Returns***
AttachDimension*(DimName)*	None.
DetachDimension*(DimName)*	None.
GetDimensionName()	Array of string.

ExplicitDimensionSet Properties

Property	***Returns***
Name	String RO
SequenceNumber	Number RO

CubeDefn Methods

Method	***Returns***
AttachDimension*(DimName)*	None
DetachDimension*(DimName)*	None
GetCauses*(CauseType)*	Array of string
GetCircularDeps*(DimName)*	Array of string
GetDimensionAggregate*(DimName)*	String
GetEffects*(EffectType)*	Array of string
GetDimensionNames()	Array of string
GetRule*(DimensionName)*	RuleDefn object
SetDimensionAggregate*(DimName,UserFunctionName)*	None
SetRule*(&RuleDefn)*	None
UsesDimension*(DimName)*	Boolean

CubeDefn Properties

Property	***Returns***
CalcAggregates	Boolean
Comments	String
DimensionCount	NumberRO
FormatType	Varies
IsVirtual	Boolean
Name	StringRO
Rule	String

Property	*Returns*
ValueDimensionName	String

CubeCollectionDefn Methods

Method	*Returns*
AttachCube*(CubeName)*	None
DetachCube*(CubeName)*	None
GetAggregateMapping*(PartName,IsCube)*	String
GetCubeNames()	Array of string
GetDimensionNames()	Array of string
GetDimSort*(DimensionName)*	Array of string
GetFieldMapping*(PartName,IsCube)*	String
GetFilter*(DimensionName)*	String
GetPersistAggregate*(DimensionName)*	String
SetAggregateMapping*(PartName,IsCube, AggregateFieldName)*	String
SetDimSort*(DimName,IsAscending*[,*CubeName1, IsAscending2*, *CubeName2*,*IsAscending3*,*CubeName3*])	None
SetFieldMapping*(PartName,IsCube,FieldName)*	None
SetFilter*(DimensionName,FilterName)*	None
SetPersistAggregate*(DimensionName,AggregateType)*	None
UsesCube*(CubeName)*	Boolean
UsesDimension*(DimensionName)*	Boolean

CubeCollectionDefn Properties

Property	*Returns*
AggregateRecName	String
Comments	Number^{RO}

Property	***Returns***
CubeCount	Number^{RO}
DimensionCount	Number^{RO}
Name	String^{RO}
RecordName	String

UserFunctionDefn Properties

Property	***Returns***
Comments	String
Name	String^{RO}

UserFunctionDefn Methods

Method	***Returns***
GetRule( )	RuleDefn object
SetRule*(&Rule)*	None

OrganizerDefn Methods

Method	***Returns***
AttachPart*(PartName,PartType)*	None
DetachPart*(PartName,PartType*	None
GetPartNames()	Array of string
UsesPart*(PartName,PartType)*	Boolean

OrganizerDefn Properties

Property	***Returns***
Comments	String
Name	String^{RO}

RuleDefn Methods

Method	***Returns***
AddRuleExpression*(&Expr)*	None
GenerateRule ( )	None

RuleDefn Property

Property	***Returns***
RuleString	String^{RO}

Assignment Method

Method	***Returns***
GenerateRule()	String

Assignment Properties

Property	***Returns***
Expression	Depends on value
Variable	String

Comparison Method

Method	***Returns***
GenerateRule ()	String

Comparison Properties

Property	***Returns***
Operand1	RuleExpression object
Operand2	RuleExpression object
Type	String ^{RO}

Constant Method

Method	***Returns***
GenerateRule()	String

Constant Properties

Property	***Returns***
Type	String RO
Value	String RO

Cube Methods

Method	***Returns***
AddIndex*(&MemberReference)*	None
GenerateRule()	String
GetIndexes()	Array of MemberReference objects

Cube Property

Property	***Returns***
Name	String RO

ExpressionBlock Methods

Method	***Returns***
AddRuleExpression*(&Expr)*	None.
GetRuleExpressions()	Array of rule expression objects.

FunctionCall Methods

Method	***Returns***
AddArgument*(&RuleExpression)*	None
GenerateRule()	String

Method	*Returns*
GetArguments()	Array of RuleExpression objects

FunctionCall Properties

Property	*Returns*
Name	String RO
Type	String

MemberReference Method

Method	*Returns*
GenerateRule()	String

MemberReference Properties

Property	*Returns*
Dimension	String RO
Member	String RO

Operation Method

Method	*Returns*
GenerateRule()	String

Operation Properties

Property	*Returns*
Operand1	RuleExpression object
Operand2	RuleExpression object
Type	String

Variable Method

Method	***Returns***
GenerateRule()	String

Variable Properties

Property	***Returns***
Name	String ^{RO}
Type	String ^{RO}

Analytic Grid Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the analytic grid classes.

Important! The analytic grid classes are not supported on IBM z/OS and Linux for IBM System z.

AnalyticGrid Function

Function	***Returns***
GetAnalyticGrid(**PAGE.***pagename*,*gridname*)	AnalyticGrid object

AnalyticGrid Methods

Method	***Returns***
GetColumn*(ColumnName)*	GridColumn object
GetCubeCollection()	CubeCollection object
LoadData()	None
SetAnalyticInstance*(AIID)*	None
SetLayout*(&SlicerArray,&RowAxisArray, &ColumnAxisArray)*	None.

AnalyticGrid Properties

Property	***Returns***
InActive	Boolean

Property	*Returns*
Label	String
ShowGridLines	Boolean
SlicerVisible	Boolean
SummaryText	String

Analytic Type Classes

This section lists the methods and properties as well as returns for analytic type classes.

Important! The analytic type classes are not supported on IBM z/OS and Linux for IBM System z.

AnalyticTypeDefn Methods

Method	*Returns*
AddModel*(ModelName,ModelType*	AnalyticTypeModelDefn object
AddRecord*(RecordName)*	AnalyticTypeRecordDefn object
Create()	
Copy*(NewAnalyticTypeName,ForceCopy)*	AnalyticTyepDefn object
Delete()	
DeleteModel*(ModelName)*	None
DeleteRecord*(RecordName)*	None
Get()	None
GetModels()	Array
GetRecordNames()	Array of string
Rename*(NewAnalyticTypeName)*	None
Save()	None

AnalyticTypeDefn Properties

Property	***Returns***
Comments	String
Description	String
Name	String^{RO}
AppClassPath	String
OwnerID	String ^{RO}

AnalyticTypeModelDefn Properties

Property	***Returns***
Name	String^{RO}
Type	String^{RO}

AnalyticTypeRecordDefn Methods

Method	***Returns***
GetSelectedFields()	Array of string
SetSelectedFields*(SelectedFieldsArray)*	None
UnsetSelectedFields*(FieldName)*	None

AnalyticTypeRecordDefn Properties

Property	***Returns***
Callback	Boolean
Description	String
Name	String^{RO}
Readable	Boolean
ReadOnce	Boolean
ScenarioManaged	Boolean
SyncOrder	Integer^{RO}

Property	*Returns*
Writeable	Boolean

Application Classes

This section lists the functions, as well as returns (if applicable) for the application classes.

Functions

Function	*Returns*
Class *classname* [{**Extends** | **Implements**}*Classname*] [Method_declarations] [Property_declarations] [**Protected** [Method_declarations] [Instance_declarations] [Constant_declaration]] [**Private** [Method_declarations] [Instance_declarations] [Constant_declaration]] **End-Class**	None
Get *PropertyName* *StatementList* **End-Get**	Depends on assignment of *StatementList*
Import *PackageName*:[*PackageName. . .*:]{*Classname* | *}	None

Function	*Returns*
Interface *classname* [{**Extends** | **Implements**} *Classname*] [Method_declarations] [Property_declarations] [**Protected** [Method_declarations] [Instance_declarations] [Constant_declaration]] **End-Interface**	None.
Method *MethodName* *StatementList* **End-Method**	Depends on method
Set *PropertyName* *StatementList* **End-Set**	None

Array

This section lists the functions, methods, and properties, as well as returns (if applicable) for the Array class.

Functions

Function	*Returns*
CreateArray*(paramlist)*	Array object
CreateArrayAny([*paramlist*] *)*	Array object
CreateArrayRept*(val,count)*	Array object
Split*(string,separator)*	Array object

Methods

Method	***Returns***
Clone()	Array object
Find*(value)*	Index of an element
Get*(index)*	Element of array
Join([*separator* [, *arraystart*, *arrayend* [, *stringsizehint*]]])	String
Next*(&index)*	Boolean
Pop()	Value of last array element
Push*(paramlist)*	None
Replace*(start,length*, [*paramlist*])	None
Reverse()	None
Set*(index)*	None
Shift()	Value of first array element
Sort([*order*])	None
Subarray*(start,length)*	Array object
Substitute*(old_val,new_val)*	None
Unshift*(paramlist)*	None

Properties

Property	***Returns***
Dimension	Number ^{RO}

Property	*Returns*
Len	Number

BPEL

This section lists the constructors, methods, and properties, as well as returns (if applicable) for the BPEL classes AsyncFFSend, BPELUtil, and IBUtil.

Constructors

Constructor	*Returns*
AsyncFFSend()	AsyncFFSend object
BPELUtil()	BPELUtil object
IBUtil()	IBUtil object

AsyncFFSend Method

Method	*Returns*
OnRequestSend*(&Msg)*	Message object

BPELUtil Methods

Method	*Returns*
GetASyncBPELProcessInstanceURL*(TransactionID)*	String
GetBPELProcessBrowserURL*(NodeName)*	String
GetOperationType*(OperationName)*	String
GetSyncBPELProcessURL*(NodeName,MessageID)*	String
LaunchASyncBPELProcess*(OperationName,&Msg, Username,Password*)	String
LaunchSyncBPELProcess*(OperationName,&Msg, Username,Password*)	Message object
UpdateConnectorResponseProperties(*&Msg)*	None

IBUtil Methods

Method	***Returns***
GetBPELConsoleURL*(NodeName)*	String
GetBPELDomain*(NodeName)*	String
GetMessageId*(TransactionID)*	String
GetNode*(MessageID)*	String
GetNumberOfRoutings*(ServiceOperationName)*	Number
GetStatus*(TransactionID)*	String
GetTransactionId*(MessageId)*	String

Charting Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the charting class.

Charting Classes Functions

Function	***Returns***
CreateObject("**Chart**")	Chart object
GetChart*(RecordName.FieldName)*	Chart object
GetChartURL*(&Chart)*	String
GetGanttChart([*Recordname.FieldName*])	Gantt chart object
GetOrgChart([*Recordname.FieldName*])	OrgChart chart object
GetRatingBoxChart([*Recordname.FieldName*])	RatingBoxChart chart object

Chart Class Methods

Method	***Returns***
Reset()	None
SetColorArray*(&Array_of_Color)*	None
SetData({*Record_Name* | *&Rowset*})	None
SetDataAnnotations*(Recordname.FieldName)*	None
SetDataColor*(Record_Name.Field_Name)*	None
SetDataGlyphScale*(Recordname.FieldName)*	None
SetDataHints*(Record_Name.Field_Name)*	None
SetDataSeries*(Record_Name.Field_Name)*	None
SetDataURLs*(URLString)*	None
SetDataXAxis*(Record_Name.Field_Name)*	None
SetDataYAxis*(Record_Name.Field_Name)*	None
SetLegend*(&Array_of_String)*	None
SetOLData({*Record_Name* | *&Rowset*})	None
SetOLDataAnnotations*(Recordname.FieldName)*	None
SetOLDataGlyphScale	None
SetOLDataSeries*(Record_Name.Field_Name)*	None
SetOLDataXAxis*(Record_Name.Field_Name)*	None
SetOLDataYAxis*(Record_Name.Field_Name)*	None
SetXAxisLabels*(&Array_of_Any)*	None
SetYAxisLabels*(&Array_of_Any)*	None

Chart Class Properties

Property	***Returns***
DataStartRow	Number
DataWidth	Number

Property	***Returns***
GridLines	String
GridLineType	String
HasLegend	Boolean
Height	Number
ImageMap	Image map ^{RO}
IsDrillable	Boolean
IsPlainImage	Boolean
IsTrueXY	Boolean
IsYAxisInteger	Boolean
LegendMaxEntries	Number
LegendPosition	String
LegendStyle	String
LineType	String
MainTitle	String
MainTitleOrient	String
MainTitleStyle	String
OLLineType	None
OLType	None
RevertToPre850	Boolean
ShowCrossHair	Boolean
Style	String
StyleSheet	String
Type	String
Width	Number
XAxisCross	Number

Property	***Returns***
XAxisCrossPoint	Number
XAxisLabelOrient	String
XAxisMax	Number
XAxisMin	Number
XAxisPrecision	Number
XAxisScaleResolution	Number
XAxisStyle	String
XAxisTicks	Number
XAxisTitle	String
XAxisTitleOrient	String
XAxisTitleStyle	String
XRotationAngle	String
YAxisCrossPoint	Number
YAxisLabelOrient	String
YAxisMax	Number
YAxisMin	Number
YAxisPrecision	Number
YAxisScaleResolution	Number
YAxisStyle	String
YAxisTicks	Number
YAxisTitle	String
YAxisTitleOrient	String
YAxisTitleStyle	String
YRotationAngle	String
ZRotationAngle	String

Gantt Methods

Method	***Returns***
Reset()	None
SetActualEndDate*(RecordName.FieldName)*	None
SetActualStartDate*(RecordName.FieldName)*	None
SetActualTaskBarColor*(Recordname.FieldName)*	None
SetChartArea*(Percentage)*	None
SetDayFormat*(Format)*	None
SetHourFormat*(Format)*	None
SetLegend*(&Array_of_String)*	None
SetMinuteFormat*(Format)*	None
SetMonthFormat*(Format)*	None
SetPlannedEndDate*(RecordName.FieldName)*	None
SetPlannedStartDate*(RecordName.FieldName)*	None
SetPlannedTaskBarColor*(RecordName.FieldName)*	None
SetSecondFormat*(Format)*	None
SetTableXScrollbar*(Scroll_ArrowAmount)*	None
SetTaskAppData*(RecordName.FieldName1*[, *RecordName.FieldName2* [,*RecordName.FieldName3. .* .]])	None
SetTaskAppDataTitles*(&TitleArray)*	None
SetTaskBarURL*(RecordName.FieldName)*	None
SetTaskData({**Record.***RecordName* | *&Rowset*})	None
SetTaskDependencyChildID*(RecordName.FieldName)*	None
SetTaskDependencyData({*RecordName* | *&Rowset*})	None
SetTaskDependencyParentID*(RecordName.FieldName)*	None

Method	***Returns***
SetTaskDependencyType *(RecordName,FieldName)*	None
SetTaskDependencyURL*(RecordName.FieldName)*	None
SetTaskDrill*(Recordname.FieldName)*	None
SetTaskExpanded*(RecordName.FieldName)*	None
SetTaskHints*(RecordName.FieldName)*	None
SetTaskID*(RecordName.FieldName)*	None
SetTaskLabel*(RecordName.FieldName)*	None
SetTaskLevel*(RecordName.FieldName)*	None
SetTaskMilestone*(RecordName.FieldName)*	None
SetTaskName*(RecordName.FieldName)*	None
SetTaskProgress*(RecordName.FieldName)*	None
SetTaskProgressBarColor*(RecordName.FieldName)*	None
SetWBSNumbering*(RecordName.FieldName)*	None
SetYearFormat*(Format)*	None

Gantt Properties

Property	***Returns***
AxisEndDateTime	Datetime
AxisStartDateTime	Datetime
DataEndDateTime	Datetime
DataStartDateTime	Datetime
DataStartRow	Number
DataWidth	Number
GridLines	String
GridLineType	String

Property	***Returns***
HasLegend	Boolean
Height	Number
ImageMap	Image map ^{RO}
InteractiveEnd	Boolean
InteractiveMove	Boolean
InteractiveProgress	Boolean
InteractiveStart	Boolean
IsDrillable	Boolean
IsPlainImage	Boolean
LegendPosition	String
LegendStyle	String
MainTitle	String
MainTitleStyle	String
PixelsPerRow	Number ^{RO}
RevertToPre850	Boolean
ShowTaskLabels	Boolean
Style	String
StyleSheet	String
TaskDependencyLineType	Number
TaskMilestoneGlyph	Number
TaskTitle	String
Width	Number
XAxisPosition	Constant
YAxisPosition	Constant

OrgChart Methods

Method	***Returns***
SetCrumbData*(&Rowset)*	None
SetCrumbRecord**(Record.***Record_Name)*	None
SetDropdownData*(&Rowset)*	None
SetDropDownRecord(**Record.***RecordName)*	None
SetLegend*(&Array_of_String)*	None
SetLegendImg*(&Array_of_String)*	None
SetNodeData*(&Rowset)*	None
SetNodeRecord**(Record.***Record_Name)*	None
SetPopUpNodeData*(&Rowset)*	None
SetPopUpNodeRecord(**Record.***Record_Name)*	None

OrgChart Properties

Property	***Returns***
CenterFocusNode	Number
ChartScrollType	Number
Collapsed_Msg	String
CollapsedImage	String
CollapseMainIconSpace	Number
CrumbDescrStyle	String
CrumbMaxDisplayLength	Number
CrumbSeparatorImage	String
DefaultImage	String
Direction	Number
DropDownBoxStyle	String

Property	*Returns*
Expanded_Msg	String
ExpandedImage	String
HasLegend	Boolean
Height	Number
ImageHeight	Number
ImageLocation	Number
ImageMouseoverMagnificationFactor	Number
LegendPosition	Number
LegendStyle	String
LegendTopSpace	Number
MainTitle	String
MainTitleStyle	String
MaxDropdownDisplayItem	Number
MaxPopupDisplayNode	Number
NodeDescr1Style	String
NodeDescr2Style	String
NodeDescr3Style	String
NodeDescr4Style	String
NodeDescr5Style	String
NodeDescr6Style	String
NodeDescr7Style	String
NodeMaxDisplayDescLength	Number
OptimizeHorizontalSpace	Number
OptimizeVerticalSpace	Number
PopupHeaderStyle	String

Property	***Returns***
PopupNodeDescr1Style	String
PopupNodeDescr2Style	String
PopupNodeDescr3Style	String
PopupNodeDescr4Style	String
PopupNodeDescr5Style	String
PopupNodeDescr6Style	String
PopupNodeDescr7Style	String
PopupNodeDescr8Style	String
Style	String
SuppressPeopleCodeOnImage	Boolean
SuppressPeopleCodeOnNode	Boolean
UnlinkCrumbDescrStyle	String
VerticalSpace	Number
Width	Number

RatingBoxChart Methods

Method	***Returns***
SetLegend*(&Array_of_String)*	None
SetLegendImg*(&Array_of_String)*	None
SetRBNodeData*(&Array_of_String)*	None
SetRBNodeRecord(**Record.***RecordName)*	None
SetXAxisLabels*(&Array_of_Any)*	None
SetYAxisLabels*(&Array_of_Any)*	None

RatingBoxChart Properties

Property	***Returns***
BoxMaxDisplayItems	Number
DraggedNodeStyle	String
GridLineType	Number
HasLegend	Boolean
Height	Number
IconOnlySelectedQuadrantStyle	String
IsDragable	Boolean
LegendPosition	Number
MainTitle	String
MainTitleStyle	String
NDMaxDisplayDescLength	Number
PopUpHeaderStyle	String
PopUpHeight	Number
PopUpStyle	String
PopUpWidth	Number
SelectedQuadrantStyle	String
ShowNodeDescription	Boolean
Style	String
ViewAllStyle	String
Width	Number
XAxisBoxNum	Number
XAxisLabelStyle	String
XAxisTitle	String
XAxisTitleStyle	String
YAxisBoxNum	Number

Property	*Returns*
YAxisLabelStyle	String
YAxisTitle	String
YAxisTitleStyle	String

Component Interface

See Appendix A, "Quick Reference for PeopleCode Classes," Component Interface Class Methods and Properties, page 2796.

Connected Query Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the connected query classes.

Connected Query Classes Constructors

Constructor	*Returns*
CONQRSMGR([*userID*], *CONQRSNAME*);	CONQRSMGR object
SEC_PROFILE()	SEC_PROFILE object
UTILITY()	UTILITY object

CONQRSMGR Methods

Method	*Returns*
CleanOutputFile()	None
Close()	None
CopyDefn*(target_UserID,target_ID)*	Boolean
DeleteDefn()	Boolean
ExistsByName *(ObjName)*	Boolean
Open*(IsNew)*	Boolean
ResetQueriesPrompt()	None

Method	***Returns***
Run*(Prompts,runProcessInfo)*	Boolean
RunToWindow()	Number
RunToXMLFormattedString*(Prompts)*	String
Save()	Boolean
SaveRunControlParms*(runCntlID)*	Boolean
SetRunControlData*(runCntlID,IsNewCtrl,IsChangeDB)*	Boolean
Validate()	Boolean
ValidateIfFieldsMapped()	Boolean
ValidateRunControlParms*(runCntlID)*	Boolean

CONQRSMGR Properties

Property	***Returns***
Const	CONQRS_CONST object ^{RO}
Description	String
Details	String
ErrString	String ^{RO}
ExecutionLog	Boolean ^{RO}
IgnoreRelFldOutput	Boolean ^{RO}
IsChanged	Boolean ^{RO}
IsDebugMode	Boolean ^{RO}
IsInit	Boolean ^{RO}
IsPublic	Boolean ^{RO}

Property	***Returns***
LastUpdatedBy	String ^{RO}
LastUpdateDTTM	Datetime ^{RO}
MaxRowsPerQuery	Number
Name	String ^{RO}
ObjectRowset	Rowset object
OprID	String ^{RO}
OutProcessFileName	String ^{RO}
PropertyArray	Array of CONQRSPROPERTY objects
QueriesPromptsArray	Array of QUERYITEMPROMPT objects ^{RO}
QueryArray	Array of string ^{RO}
RegisteredBy	String ^{RO}
RegisteredDTTM	Datetime ^{RO}
RunCntlId	String ^{RO}
RunControlParArray	Array of PSCONQRSRUNPRM records ^{RO}
RunMode	Number ^{RO}
SchedInfo	SCHED_INFO object
SchedRequestType	String
SecProfile	SEC_PROFILE object ^{RO}

Property	*Returns*
ShowFormattedXML	Boolean RO
Status	String
XMLDataFileName	String RO
XMLDataFullName	String RO

CONQRSPROPERTY Properties

Property	*Returns*
PROPDEFAULT	String
PROPNAME	String
PROPVALUE	String
PROPVALUEARRAY	String
PROPVALUEARRAYDESC	String

CONQRS_CONST Properties

Property	*Returns*
InitExisting	Boolean RO
InitNew	Boolean RO
MsgSet	Number RO
RunCntlId_Auto	String RO
RunMode_Prev	Number RO

Property	Returns
RunMode_Prev_DefRowNumber	Number RO
RunMode_Sample	Number RO
RunMode_Sched_File	Number RO
RunMode_Sched_Web	Number RO
RunMode_Window	Number RO
RunMode_XMLFormattedString	Number RO
SchedRequest_CQR	String RO
SchedRequest_XMLP	String RO
Stat_Active	String RO
Stat_InActive	String RO
Stat_InProgress	String RO

QUERYITEMPROMPT Properties

Property	Returns
QueryName	String RO
QueryPromptRecord	Record object RO

SCHED_INFO Properties

Property	Returns
AE_ID	String

Property	Returns
DIRLOCATION	String
OPRID	String
OUTDESTTYPE	Number
PRCSFILENAME	String
PROCESS_INSTANCE	String
RUN_CNTL_ID	String

SEC_PROFILE Properties

Property	Returns
CanCreatePublic	Boolean ^{RO}
CanModify	Boolean ^{RO}
CanRunQuery	Boolean ^{RO}

UTILITY Methods

Method	Returns
CheckQryForTreePrompt*(QryName,scope)*	String
CheckQrySecurity*(QryName,IsPublicObject)*	Boolean
GetQueryScopeByName*(QryName)*	Number
ValidateObjectID*(ID)*	String

Crypt Class

This section lists the methods and properties, as well as returns (if applicable) for the Crypt class.

Function

Function	***Returns***
CreateObject("Crypt")	Crypt object

Methods

Method	***Returns***
FirstStep()	None
GoToStep*(StepNumber)*	None
LoadLibrary*(LibraryFile,LibraryID)*	None
NextStep()	None
Open*(ProfileName)*	None
SetParameter*(Name,Value)*	None
UpdateData*(Data)*	None

Properties

Property	***Returns***
Result	String RO
Verified	Boolean RO

Document Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the connected query classes.

Document Classes Functions

Function	***Returns***
CreateDocument*(DocumentKey* | *Package, DocumentName,Version*)	Document object
CreateDocumentKey*(Package,DocumentName,Version)*	DocumentKey object

Document Methods

Method	***Returns***
GenXmlString([*validate*])	String
GetDocumentKey()	DocumentKey object
GetElement*(ElementName)*	Collection object, Compound object, or Primitive object
GetRowset()	Rowset object
GetSchema()	String
ParseXmlFromFile*(file_name* [, *validate*])	Boolean
ParseXmlFromURL*(URL* [, *validate*])	Boolean
ParseXmlString(*XML_string* [, *validate*])	Boolean
UpdateFromRowset*(&Doc_RS)*	Boolean
ValidateData()	Boolean

Document Properties

Property	***Returns***
DocumentElement	Compound object RO

DocumentKey Methods

Method	***Returns***
SetDocumentKey*(Package,DocumentName,Version)*	Boolean

DocumentKey Properties

Property	***Returns***
DocumentName	String RO
PackageName	String RO

Property	***Returns***
Version	String RO

Primitive Methods

Method	***Returns***
GetEnumName*(index)*	String
GetParent()	Collection object or Compound object
GetPath()	String

Primitive Properties

Property	***Returns***
ElementType	Integer RO
EnumCount	Integer RO
IsChanged	Boolean RO
IsInitialized	Boolean RO
IsRequired	Boolean RO
MaxDefinedDecimalLength	Integer RO
MaxDefinedLength	Integer RO
Name	String RO
OrigValue	String
PrimitiveSubType	Integer RO
PrimitiveType	Integer RO

Property	***Returns***
SequenceNumber	Integer ^{RO}
Value	String

Compound Methods

Method	***Returns***
GetParent()	Collection object or Compound object
GetPath()	String
GetPropertyByIndex*(index)*	Collection object, Compound object, or Primitive object
GetPropertyByName*(prop_name)*	Collection object, Compound object, or Primitive object
GetUniqueKey()	Array of string

Compound Properties

Property	***Returns***
ElementType	Integer ^{RO}
IsChanged	Boolean ^{RO}
IsInitialized	Boolean ^{RO}
IsRequired	Boolean ^{RO}
Name	String ^{RO}
PropertyCount	Integer ^{RO}
SequenceNumber	Integer ^{RO}

Collection Methods

Method	***Returns***
AppendItem*(&Elem)*	Boolean
CreateItem()	Collection object, Compound object, or Primitive object
DeleteItem*(index)*	Boolean
GetItem*(index)*	Collection object, Compound object, or Primitive object
GetParent()	Collection object or Compound object
GetPath()	String
InsertItem*(&Elem,index)*	Boolean

Collection Properties

Property	***Returns***
CollectionElementType	Integer RO
Count	Integer RO
DefinedMaxOccurs	Integer RO
DefinedMinOccurs	Integer RO
ElementType	Integer RO
IsChanged	Boolean RO
IsInitialized	Boolean RO
IsRequired	Boolean RO
Name	String RO
SequenceNumber	Integer RO

Exception Class

This section lists the functions, methods, and properties, as well as returns (if applicable) for the exception class.

Functions

Function	***Returns***
CreateException*(message_set, message_num,default_txt* [, *subslist*])	Exception object
Throw *Expression*	None
try *Protected StatementList* catch *QualifiedID &Id* *StatementList* end-try	None

Methods

Method	***Returns***
GetSubstitution*(Index)*	String
Output()	String
SetSubstitution*(Index,String)*	None
ToString()	String

Properties

Property	***Returns***
Context	String RO
DefaultText	String
MessageNumber	Number RO
MessageSetNumber	Number RO
MessageSeverity	String RO

Property	Returns
StackTrace	String RO
SubstitutionCount	Number RO

Feed Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the feed classes.

Feed Classes Constructors

Constructor	Returns
FeedFactory.createFeed("*feed_ID*")	Feed object
FeedFactory()	FeedFactory object
FeedFactory.getDataSource("*DS_ID*")	DataSource object
DataSourceParameter("*DSP_ID*", *&Parent_DS*)	DataSourceParameter object
DataSourceParameterValue("*ID*", *&Parent_DSP*)	DataSourceParameterValue object
DataSourceSetting("*DSS_ID*", *&Parent_DS*)	DataSourceSetting object
Utility()	Utility object
FeedFactory.Utility.getFeedDoc ("*feed_ID*", *Format, &Attributes*)	FeedDoc object
FeedDoc.addEntry("*entry_ID*")	FeedEntry object

Feed Methods

Method	Returns
delete()	Boolean
equals*(&Object)*	Boolean
execute()	FeedDoc object
getAttribute*(attribute_ID)*	Attribute object
load()	Boolean
populatePrefData*(&FeedRequest)*	None

Method	***Returns***
publishToSites*(Sites)*	None
resetFeedAttributes()	FeedAttributes collection
resetFeedSecurities()	FeedSecurities collection
save()	Boolean
saveAs*(new_ID,CopyPrefData)*	Feed object
setAttribute*(attribute_ID,Value,XML)*	Attribute object
setDataSourceById*(DataType_ID)*	DataSource object
unpublishFromSites*(Sites)*	None

Feed Properties

Property	***Returns***
Authorized	Boolean [RO]
CategoryID	String
CreateDTTM	Datetime [RO]
CreateNode	String
CreateOprID	String [RO]
CreatePortal	String
DataSource	DataSource object
DataTypeID	String [RO]
Description	String
FeedAttributes	FeedAttributes collection

Property	***Returns***
FeedAuthorizationOprID	String
FeedAuthorizationOprPWD	String
FeedAuthorizationType	String
FeedCacheTime	*not current used*
FeedCacheType	*not current used*
FeedContentUrl	String [RO]
FeedFactory	FeedFactory object
FeedFormat	String
FeedSecurityType	String
FeedUrl	String [RO]
HasAdminParams	Boolean [RO]
HasUserParams	Boolean [RO]
IBOperationName	String
ID	String [RO]
LastPubDTTM	Datetime [RO]
LastUpdDTTM	Datetime [RO]
LastUpdOprID	String [RO]
NameSpaceID	String

Property	***Returns***
ObjectType	String [RO]
OperatingMode	Number [RO]
OwnerID	String
PublishedInSites	Array of string [RO]
Title	String
Utility	Utility object

FeedFactory Methods

Method	***Returns***
convertFeedLinksToHoverMenu*(&Links,Flat)*	HoverMenu object
convertFeedLinksToOPML*(&Links,Flat)*	OPMLDoc object
createFeed*(feed_ID)*	Feed object
deleteFeed*(feed_ID)*	Boolean
genFeedUrl*(feed_ID,OperationName,SecurityType)*	String
genUniqueFeedId*(seed)*	String
getAllPagedFeedLinks*(feed_ID,fromDTTM, allLanguages)*	Array of string
getCategory*(category_ID,ActiveOnly)*	Array of string
getDataSource*(DataType_ID)*	DataSource object
getFeed*(feed_ID,mode)*	Feed object
getFeedDoc*(&feedRequest)*	FeedDoc object
getFeedLink*(feed_ID)*	Link object
getFeedLinks*(&searchRequest)*	Link collection

Method	***Returns***
getRelatedFeedsHoverMenu*(&Requests)*	HoverMenu object
getRelativePageLinkForFeed*(feed_ID,currentFeedURL, linkOption,fromDTTM*)	String

FeedFactory Properties

Property	***Returns***
DataSources	DataSource collection RO
Feeds	Feed collection RO
Utility	Utility object

DataSource Methods

Method	***Returns***
addParameter*(DSparam_ID,Value)*	DataSourceParameter object
equals*(&Object)*	Boolean
getAttributeById*(attribute_ID)*	Attribute object
getContentUrl()	String
getDataSecurity()	Authorization objects
getParameterById*(DSP_ID)*	DataSourceParameter object
getParameterDetail()	String
getSettingDetail()	String
isCurrentUserAdmin()	Boolean
isCurrentUserAuthorized()	Boolean
onDelete()	None
resetParameters()	DataSourceParameter collection

DataSource Properties

Property	***Returns***
AdminPersonalizationPage	PSComponent object
AllowCustomParameters	Boolean RO
AllowRealTimeFeedSecurity	Boolean RO
DataSourceType	String RO
DefaultFeedAttributes	Attribute collection
DefaultIBOperationName	String RO
Description	String
HasParameters	Boolean RO
HasSettings	Boolean RO
IBOperations	IBOperation collection
ID	String RO
LongDescription	String
ObjectType	String RO
Parameters	DataSourceParameter collection RO
ParametersCompleted	Boolean RO
Parent	Feed object
PortalSpecificPersonalization	Boolean RO
Settings	DataSourceSetting collection RO

Property	*Returns*
SettingsCompleted	Boolean RO
UserPersonalizationPage	PSComponent object
Utility	Utility object

DataSourceParameter Methods

Method	*Returns*
addUserValue*(DSPValue_ID,Value)*	DataSourceParameterValue object
clone()	DataSourceParameter object
equals*(&Object)*	Boolean
resetUserValues()	UserValues collection
setRangeFromFieldTranslates*(FieldName)*	None
validateValue*(Value,CheckPrompt)*	String

DataSourceParameter Properties

Property	*Returns*
AllowChangesToRequired	Boolean
DefaultValue	String
DefaultValueForDisplay	String RO
Description	String
EditType	Number
EvaluatedValue	String RO
FieldType	Number

Property	***Returns***
ID	String
Name	String
ObjectType	String [RO]
Parent	DataSource object
PromptTable	String
Range	Array of string
Required	Boolean
SkipValidityChecks	Boolean
UsageType	String
UserValues	UserValues collection [RO]
Utility	Utility object
Value	String
ValueForDisplay	String [RO]

DataSourceParameterValue Methods

Method	***Returns***
clone()	DataSourceParameterValue object
equals*(&Object)*	Boolean

DataSourceParameterValue Properties

Property	***Returns***
Description	String
ID	String ^{RO}
Name	String
ObjectType	String ^{RO}
OrderNumber	Number
Parent	DataSourceParameter object
Value	String

DataSourceSetting Methods

Method	***Returns***
clone()	DataSourceSetting object
equals*(&Object)*	Boolean
setRangeFromFieldTranslates*(FieldName)*	None

DataSourceSetting Properties

Property	***Returns***
DefaultValue	String
DisplayOnly	Boolean ^{RO}
EditType	Number
Enabled	Boolean
FieldType	Number

Property	*Returns*
ID	String RO
LongName	String
Name	String
ObjectType	String RO
Parent	DataSource object
PromptTable	String
Range	Array of string RO
RefreshOnChange	Boolean
RelatedFieldValue	String
Required	Boolean
ShowRelatedField	Boolean RO
Utility	Utility object
Value	String
Visible	Boolean

Utility Methods

Method	*Returns*
dateStringToUserPref*(DateString)*	String
datetimeToString*(Datetime)*	String
decodeXML*(String)*	String

Method	***Returns***
encodeXML*(String)*	String
evaluateSysVar*(SysVar)*	String
genNameSpaceID*(NameSpace_ID)*	String
getExceptionText*(&Exception)*	String
getFeedDoc*(feed_ID,Format,&Attributes)*	FeedDoc object
getFeedMimeType*(Format)*	String
getFieldTranslates*(FieldName)*	Array of array of string
getNodeValue*(String,Tag)*	String
getUserDateFormat()	String
getUserInfo*(user_ID)*	Array of string
httpStringToDatetime*(string)*	Datetime
join*(&Array,Tag)*	String
join2D*(&Array,Tag,ChildTags)*	String
setMessageHeadersAndMimeType*(&Message,&feeddoc, &FeedRequest)*	Message object
setNodeValue*(Value,Tag)*	String
showException*(Exception)*	None
showInvalidValueException*(Name,Value)*	None
split()	Array of string
split2D()	Array of array of string
stringToDate*(DateString)*	Date
stringToDatetime*(DatetimeString)*	Datetime
validateSysVar*(SysVar)*	String
viewStringAsAttachment*(String,FileName, ViewAttachment)*	None

Utility Properties

Property	***Returns***
ATTACHMENT_URL	String RO
AUTHTYPE_PERM	String RO
AUTHTYPE_ROLE	String RO
DSPARAMETER_INCREMENTAL	String RO
DSPARAMETER_MAXROW	String RO
DSPARAMETER_SF_MAXMINUTES	String RO
DSPARAMETER_SF_PAGING	String RO
EDITTYPE_NOTABLEEDIT	Integer RO
EDITTYPE_PROMPTTABLE	Integer RO
EDITTYPE_TRANSLATETABLE	Integer RO
EDITTYPE_YESNO	Integer RO
FEEDATTRIBUTE_AUTHOR	String RO
FEEDATTRIBUTE_CLOUD	String RO
FEEDATTRIBUTE_COMPLETE	String RO
FEEDATTRIBUTE_CONTRIBUTOR	String RO
FEEDATTRIBUTE_COPYRIGHT	String RO
FEEDATTRIBUTE_EXPIRES	String RO

Property	***Returns***
FEEDATTRIBUTE_ICONURL	String RO
FEEDATTRIBUTE_LOGOURL	String RO
FEEDATTRIBUTE_MANAGINGEDITOR	String RO
FEEDATTRIBUTE_MAXAGE	String RO
FEEDATTRIBUTE_PERSINSTRUCTION	String RO
FEEDATTRIBUTE_RATING	String RO
FEEDATTRIBUTE_SKIPDAYS	String RO
FEEDATTRIBUTE_SKIPHOURS	String RO
FEEDATTRIBUTE_TEXTINPUT	String RO
FEEDATTRIBUTE_TTL	String RO
FEEDATTRIBUTE_WEBMASTER	String RO
FEEDATTRIBUTE_XSL	String RO
FEEDAUTHTYPE_ALL	String RO
FEEDAUTHTYPE_ANONYMOUS	String RO
FEEDAUTHTYPE_DEFAULT	String RO
FEEDCACHETYPE_NONE	String RO
FEEDCACHETYPE_PRIVATE	String RO
FEEDCACHETYPE_PUBLIC	String RO

Property	*Returns*
FEEDCACHETYPE_ROLE	String RO
FEEDFORMAT_ATOM10	String RO
FEEDSECUTYPE_PUBLIC	String RO
FEEDSECUTYPE_REALTIME	String RO
FEEDSECUTYPE_SELECTED	String RO
FEEDTYPE_DYNAMIC	String RO
FEEDTYPE_PREPUBLISHED	String RO
FIELDTYPE_CHARACTER	Integer RO
FIELDTYPE_DATE	Integer RO
FIELDTYPE_DATETIME	Integer RO
FIELDTYPE_LONGCHARACTER	Integer RO
FIELDTYPE_NUMBER	Integer RO
FIELDTYPE_SIGNEDNUMBER	Integer RO
FIELDTYPE_TIME	Integer RO
IBSOTYPE_ASYNC	String RO
IBSOTYPE_SYNC	String RO
IBSOTYPE_UNKNOWN	String RO
ICONURL_FEED_A	String RO

Property	***Returns***
ICONURL_FEED_IA	String ^{RO}
INCREMENTALOPTION_NO	String ^{RO}
INCREMENTALOPTION_YES	String ^{RO}
LINKTYPE_FIRST	String ^{RO}
LINKTYPE_LAST	String ^{RO}
LINKTYPE_NEXT	String ^{RO}
LINKTYPE_PREVIOUS	String ^{RO}
MIMETYPE_ATOM	String ^{RO}
MIMETYPE_OPML	String ^{RO}
MIMETYPE_XML	String ^{RO}
OPERATINGMODE_AUTHORIZATION	Integer ^{RO}
OPERATINGMODE_DEFAULT	Integer ^{RO}
OPERATINGMODE_DELETION	Integer ^{RO}
OPERATINGMODE_EXECUTION	Integer ^{RO}
OPERATINGMODE_EXECUTION_NOENTRY	Integer ^{RO}
QUERYPARAMETER_CHILDFEEDID	String ^{RO}
QUERYPARAMETER_DATATYPEID	String ^{RO}
QUERYPARAMETER_DEFLOCALNODE	String ^{RO}

Property	***Returns***
QUERYPARAMETER_DSSCOUNT	String RO
QUERYPARAMETER_DSSNAME	String RO
QUERYPARAMETER_DSSVALUE	String RO
QUERYPARAMETER_FEEDFORMAT	String RO
QUERYPARAMETER_FEEDID	String RO
QUERYPARAMETER_FEEDLIST	String RO
QUERYPARAMETER_FEEDTYPE	String RO
QUERYPARAMETER_IBTRANSID	String RO
QUERYPARAMETER_IFMODIFIEDSINCE	String RO
QUERYPARAMETER_IFNONEMATCH	String RO
QUERYPARAMETER_KEYWORD	String RO
QUERYPARAMETER_LANGUAGE	String RO
QUERYPARAMETER_NODENAME	String RO
QUERYPARAMETER_PAGENUM	String RO
QUERYPARAMETER_PORTALNAME	String RO
QUERYPARAMETER_PTPPB_SEARCH_MODE	String RO
QUERYPARAMETER_PTPPB_SEARCH_TEXT	String RO
RequestInfo	FeedRequest object

Property	*Returns*
SF_MAXMINUTES_ALLMSGS	Integer RO
SF_MAXROWOPTION_ALLMSGS	Integer RO
SF_MAXROWOPTION_LATESTMSG	Integer RO
SF_PAGINGOPTION_NOPAGING	Integer RO
SF_PAGINGOPTION_SEGMENTED	Integer RO
SYSVAR_INVALID	String RO
USAGETYPE_ADMINSPECIFIED	String RO
USAGETYPE_FIXED	String RO
USAGETYPE_INTERNAL	String RO
USAGETYPE_NOTUSED	String RO
USAGETYPE_SYSVAR	String RO
USAGETYPE_USERSPECIFIED	String RO
XMLCHILDELEMENTS_CLOUD	Array of string RO
XMLCHILDELEMENTS_PERSON	Array of string RO
XMLCHILDELEMENTS_TEXTINPUT	Array of string RO
XMLELEMENT_DAY	String RO
XMLELEMENT_DESCRIPTION	String RO
XMLELEMENT_DOMAIN	String RO

Property	*Returns*
XMLELEMENT_EMAIL	String RO
XMLELEMENT_HOUR	String RO
XMLELEMENT_LINK	String RO
XMLELEMENT_NAME	String RO
XMLELEMENT_PATH	String RO
XMLELEMENT_PORT	String RO
XMLELEMENT_PROTOCOL	String RO
XMLELEMENT_REGISTERPROCEDURE	String RO
XMLELEMENT_TITLE	String RO

FeedDoc Methods

Method	*Returns*
addCategory*(category)*	Boolean
addEntry*(entry_ID)*	FeedEntry object
datetimeToString*(Datetime)*	String
deleteCategory*(category)*	Boolean
equals*(&Object)*	Boolean
getEntry*(entry_ID)*	FeedEntry object
resetEntries()	FeedEntry collection
stringToDatetime*(DatetimeString)*	Datetime

FeedDoc Properties

Property	***Returns***
AllowMoreEntries	Boolean RO
ApplicationRelease	String RO
Categories	Array of string RO
ContentUrl	String
Copyright	String
Description	String
Entries	FeedEntry collection RO
Expires	Datetime
FeedFormat	String RO
FeedUrl	String
FirstUrl	String
Generator	String
ID	String RO
LastUrl	String
Logo	String
MaxAge	Number
MaxEntries	Number
NextUrl	String

Property	**Returns**
ObjectType	String RO
PreviousUrl	String
RootElement	XmlNode object RO
Title	String
Updated	Datetime

FeedEntry Methods

Method	**Returns**
addCategory*(category)*	Boolean
addContributor*(name,email)*	Boolean
addEnclosure*(URL,type,length)*	Boolean
delete()	None
deleteCategory*(category)*	Boolean
deleteContributor*(name)*	Boolean
deleteEnclosure*(URL)*	Boolean
equals*(&Object)*	Boolean

FeedEntry Properties

Property	**Returns**
Author	Array of string
Categories	Array of string RO
Comments	String

Property	***Returns***
ContentUrl	String
Contributors	Array of array of string ^{RO}
Copyright	String
Description	String
Enclosures	Array of array of string ^{RO}
Expires	Datetime
FeedDoc	FeedDoc object ^{RO}
FullContent	String
GUID	String
ID	String ^{RO}
MaxAge	Number
ObjectType	String ^{RO}
Published	Datetime
Title	String
Updated	Datetime

Field

This section lists the functions, methods, and properties, as well as returns (if applicable) for the field class.

Function

Function	***Returns***
GetField([*recname.fieldname*])	Field object

Methods

Method	***Returns***
AddDropDownItem*(CodeString,DescriptionString)*	None
ClearDropDownList()	None
EncryptPETKey()	None
DecryptPETKey()	None
GetAuxFlag*(FlagNumber)*	Boolean
GetLongLabel*(LabelID)*	String
GetRelated*(recname.fieldname)*	Field object
GetShortLabel*(LabelID)*	String
SearchClear()	None
SetCursorPos({**PAGE.***pagename* | **%Page**})	None
SetDefault()	None

Properties

Property	***Returns***
DataAreaCollapsed	Boolean
DecimalPosition	Number

Property	***Returns***
DisplayFormat	String
DisplayOnly	Boolean
DisplayZero	Boolean
DisplayZeroChanged	Boolean
EditError	Boolean
Enabled	Boolean
FieldLength	Number RO
FormatLength	Number RO
FormattedValue	None
HoverText	String
IsAltKey	Boolean RO
IsAuditFieldAdd	Boolean RO
IsAuditFieldChg	Boolean RO
IsAuditFieldDel	Boolean RO
IsAutoUpdate	Boolean RO
IsChanged	Boolean RO
IsDateRangeEdit	Boolean RO
IsDescKey	Boolean RO

Property	*Returns*
IsDuplKey	Boolean RO
IsEditTable	Boolean RO
IsEditXlat	Boolean RO
IsFromSearchField	Boolean RO
IsInBuf	Boolean RO
IsKey	Boolean RO
IsListItem	Boolean RO
IsNotUsed	Boolean RO
IsRequired	Boolean RO
IsRichTextEnabled	Boolean RO
IsSearchItem	Boolean RO
IsSystem	Boolean RO
IsThroughSearchField	Boolean RO
IsUseDefaultLabel	Boolean RO
IsYesNo	Boolean RO
Label	String
LabelImage	{**Image.***imagename* | String}
LongTranslateValue	Any

Property	***Returns***
MessageNumber	Number RO
MessageSetNumber	Number RO
MouseOverMsgNum	Number
MouseOverMsgSet	Number
Name	String RO
OriginalValue	Depends on field
ParentRecord	Record Object RO
PromptTableName	String RO
SearchDefault	Boolean
SearchEdit	Boolean
SetComponentChanged	Boolean
ShortTranslateValue	Any
ShowRequiredFieldCue	Boolean
SmartZero	Boolean
SqlText	String
StoredFormat	String RO
Style	String
Type	String RO

Property	***Returns***
Value	Depends on field
Visible	Boolean

File

This section lists the functions, methods, and properties, as well as returns (if applicable) for the file class.

Functions

Function	***Returns***
CreateDirectory*(path*, [, *pathtype*])	None
FileExists*(filename* [, *pathtype*])	Boolean
FindFiles(*filespec_pattern* [, *pathtype*])	Array object
GetFile*(filename,mode* [, *charset*] [,*pathtype*])	File object
GetTempFile(*filename,mode* [, *charset*] [,*pathtype*])	File object

Methods

Method	***Returns***
Close()	None
CreateRowset()	Rowset object
GetPosition()	Number
GetString([*Strip_Line_Terminator*])	String
Open*(filename,mode* [, *charset*] [, *pathtype*])	None

Method	***Returns***
ReadLine(*string)*	Boolean
ReadRowset()	Rowset object
SetFileId*(fileid,position)*	
SetFileLayout(FILELAYOUT.*filelayoutname)*	Boolean
SetPosition*(position)*	None
SetRecTerminator*(Terminator)*	None
WriteLine*(string)*	None
WriteRaw*(RawBinary)*	None
WriteRecord*(record)*	Boolean
WriteRowset*(rowset* [, *Write_Data*])	Boolean
WriteString*(string)*	None

Properties

Property	***Returns***
CurrentRecord	String ^{RO}
IgnoreInvalidId	Boolean
IsError	Boolean ^{RO}
IsNewFileId	Boolean ^{RO}
IsOpen	Boolean ^{RO}

Property	*Returns*
Name	String [RO]
TerminateLines	Boolean
UseSpaceForNull	Boolean
ZeroExtend	Boolean

Grid Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the grid classes.

Grid Functions

Function	*Returns*
GetGrid(**PAGE.***pagename*,*gridname* [, *occursnumber*])	Grid object

Grid Methods

Method	*Returns*
GetColumn*(columnname)*[D]	Grid column object
EnableColumns*(&Array)*	None
LabelColumns*(&Array)*	None
SetProperties*(&Array)*	None
ShowColumns*(&Array)*	None

Grid Properties

Property	***Returns***
gridcolumnname	Grid column object
Label	String
SummaryText	String

GridColumn Properties

Property	***Returns***
Enabled	Boolean
Label	String
Name	String RO
Visible	Boolean

Internet Script Classes

This section lists the system variables, methods, and properties, as well as returns (if applicable) for the internet script (iScript) classes.

Internet Script System Variables

System Variable	***Returns***
%Request	Request object
%Response	Response object

Request Methods

Method	***Returns***
GetCookieNames()	Array of string
GetCookieValue*(name)*	String

Method	***Returns***
GetHeader*(name)*	String
GetHeaderNames()	Array of string
GetHelpURL*(HelpContext)*	String
GetParameter*(name)*	String
GetParameterNames()	Array of string
GetParameterValues*(Name)*	Array of string

Request Properties

Property	***Returns***
AuthTokenDomain	String RO
AuthType	String RO
BrowserPlatform	String RO
BrowserType	String RO
BrowserVersion	String RO
ByPassSignOn	Boolean RO
ContentURI	String RO
ExpireMeta	String RO
FullURI	String RO
HTTPMethod	String RO

Property	***Returns***
LogoutURL	String RO
PathInfo	String RO
Protocol	String RO
QueryString	String RO
RelativeURL	String RO
RemoteAddr	String RO
RemoteHost	String RO
RequestURI	String RO
RemoteUser	String RO
Scheme	String RO
ServerName	String RO
ServerPort	String RO
ServletPath	String RO
Timeout	Integer RO

Response Methods

Method	***Returns***
Clear()	None
CreateCookie*(name)*	Cookie object

Method	***Returns***
GetCookie*(name)*	Cookie object
GetCookieNames()	Array of string
GetHeader*(name)*	String
GetHeaderNames()	Array of string
GetImageURL*(ImageName)*	String
GetJavaScriptURL(HTML.*Name)*	Srting
GetStyleSheetURL(STYLESHEET.*name*)	String
RedirectURL*(name)*	None
SetContentType*(Type)*	None
SetHeader*(name,value)*	None
UseSimpleURL({*True* | *False*})	None
Write*(HTML)*	None
WriteLine*(HTML)*	None

Response Properties

Property	***Returns***
Charset	String ^{RO}
DefaultStyleSheetName	String ^{RO}

Cookie Properties

Property	*Returns*
Domain	String
MaxAge	Number
Name	String RO
Path	String
Secure	Boolean
Value	String

Java

This section lists the functions for the Java class.

Functions

Function	*Returns*
CopyFromJavaArray(*JavaArray*,*&PeopleCodeArray* [, *&RestrictionArray*])	None.
CopyToJavaArray(*&PeopleCodeArray*,*JavaArray* [, *&RestrictionArray*])	None.
CreateJavaArray(*ElementClassName[], NumberOfElements)*	Java object
CreateJavaObject(*ClassName* [*ConstructorParams*])	Java object
GetJavaClass(*ClassName)*	Java object

Mail Classes

This section lists the methods and properties as well as returns (if applicable) for the mail (MultiChannel Framework) classes.

Constructors

Constructor	***Returns***
MCFBodyPart()	MCFBodyPart object
MCFGetMail()	MCFGetMail object
MCFOutboundEmail()	MCFOutboundEmail object
MCFMailStore()	MCFMailStore object
SMTPSession()	SMTPSession object

MCFBodyPart Methods

Method	***Returns***
AddHeader*(HeaderName,HeaderValue)*	None.
GetHeader*(HeaderName)*	String
GetHeaderCount()	Number
GetHeaderName*(Index)*	String
GetHeaderNames()	String
GetHeaderValues*(Index)*	String
GetUnparsedHeaders()	String
SetAttachmentContent({*FilePath* | *FileURL*}, *FilePathType*,*FileName*, *FileDescr*,*OverrideContentType*, *OverrideCharset)*	None

MCFBodyPart Properties

Property	***Returns***
AttachmentURL	String
Charset	String
ContentType	String
Description	String

Property	***Returns***
Disposition	String
Filename	String
FilePath	String
FilePathType	String
IsAttachment	Boolean ^{RO}
MultiPart	String
Text	String

MCFEmail Properties

Property	***Returns***
BCC	String
BounceTo	String
CC	String
From	String
Importance	String
Priority	Number
Recipients	String
RefIDs	String
ReplyIDs	String
ReplyTo	String
Sender	String
Sensitivity	String
Subject	String
Text	String

MCFGetMail Methods

Method	***Returns***
CreateQuarantineFolder ( )	Boolean
GetCount ( )	Number
GetEmailCount*(user,password,server,node)*	Number
ReadAllEmailHeadersWithAttach*(user,password, server,node*)	Rowset object
ReadEmails *(Count)*	String
ReadEmailsWithAttach*(User,password,server,node, count)*	Rowset object
ReadEmailsWithUID *(UID)*	Array of MCFInboundEmail objects
ReadEmailWithAttach*(User,password,server,node, UID)*	Rowset object
ReadHeaders ( )	Array of objects
RemoveEmail*(user,password,server,node,UID list)*	Boolean
RemoveEmails*(UID_List)*	String
SetMCFEmail *(user,password,server,node)*	None

MCFGetMail Properties

Property	***Returns***
AttachmentRoot	String
ContentTypes	String
ErrorCount	Number ^{RO}
IBNode	String
MailServer	String

Property	***Returns***
Password	String
QuarantineCount	Number RO
QuarantineFolder	String
Status	String RO
UserID	String

MCFInboundEmail Methods

Method	***Returns***
DumpToFile *(&File)*	None
GetAttachments ( )	Array of objects
GetFrom ( )	Array of string
GetParts ( )	Array of objects
GetSender ( )	Array of string
ReadFromDatabase *(Email_ID)*	Boolean
SaveToDatabase ( )	Number

MCFInboundEmail Properties

Property	***Returns***
AttachList	String
AttachSizes	String
DttmReceived	String
DttmSaved	String
DttmSent	String
IBNode	String
Language	String
MessageID	String

Property	*Returns*
NotifyCC	String
NotifyTo	String
OffsetReceived	Number
OffsetSent	String
Server	String
Size	Number
Status	String RO
UID	String
User	String

MCFMailStore Methods

Method	*Returns*
AuthorizeEmailAttach*(email ID,authentication name, authentication type, operation)*	Boolean
DeleteEmail*(email ID,* [*forced delete*])	Boolean
RetrieveEmail*(email ID)*	Rowset object
StoreEmail*(rowset,row)*	Number

MCFMailUtil Methods

Method	*Returns*
DecodeText *(TextToDecode,&DecodedText)*	Boolean
DecodeWord *(WordToDecode,&DecodedWord)*	Boolean
EncodeText *(TextToEncode,charset,EncodingStyle, &EncodedText*)	Boolean
EncodeWord *(WordToEncode,charset,EncodingStyle, &EncodedWord*)	Boolean

Method	***Returns***
GetErrorMsgParam(*&index)*	String
IsDomainNameValid*(domainname)*	Boolean
IsEmailServerAvailable *(server,port,user,password)*	Boolean
ParseRichTextHtml*(richtext)*	Array of MCFBodyPart objects
ValidateAddress *(addresslist)*	Boolean

MCFMailUtil Properties

Property	***Returns***
badaddresses	Array of string[RO]
ErrorDescription	String [RO]
ErrorDetails	String[RO]
ErrorMsgParamsCount	Integer[RO]
imagesLocation	String
MessageNumber	Number [RO]
MessageSetNumber	Number [RO]

MCFMultiPart Methods

Method	***Returns***
AddBodyPart *(&bodyPart)*	None
GetBodyPart*(Index)*	MCFBodyPart object
GetContentType ( )	String
GetCount ( )	Number

MCFMultiPart Property

Property	***Returns***
SubType	String

MCFOutboundEmail Methods

Method	***Returns***
AddAttachment({*FilePath* | *FileURL*}, *FilePathType, FileName, FileDescr,OverrideContentType, OverrideCharset* [,*UploadPageTitle*])	None
AddHeader*(HeaderName,HeaderValue)*	None
GetErrorMsgParam*(&index)*	String
GetHeader*(HeaderName)*	Array of string
GetHeaderCount ()	Integer
GetHeaderName*(Index)*	String
GetHeaderNames ( )	String
GetHeaderValues *(Index)*	String
Send ( )	Number
SetSMTPParam*(ParamName,ParamValue)*	None

MCFOutboundEmail Properties

Property	***Returns***
BackupSMTPPort	Number
BackupSMTPServer	String
BackupSMTPSSLClientCertAlias	String
BackupSMTPSSLPort	Number
BackupSMTPUserName	String
BackupSMTPUseSSL	Boolean
BCC	String
BounceTo	String
CC	String
Charset	String

Property	***Returns***
ContentLanguage	String
ContentType	String
DefaultCharSet	String
Description	String
Disposition	String
ErrorDescription	String ^{RO}
ErrorDetails	String
ErrorMsgParamsCount	Integer^{RO}
Filename	String
FilePath	String
FilePathType	String
Importance	String
InvalidAddresses	String ^{RO}
IsAuthenticationReqd	Boolean
IsOkToSendPartial	Boolean
IsReturnReceiptReqd	Boolean
MessageNumber	Number ^{RO}
MessageSetNumber	Number ^{RO}
MultiPart	String
Priority	Number
Recipients	String
RefIDs	Number
ReplyIDs	Number
ReplyTo	String
ResultOfSend	Number ^{RO}

Property	*Returns*
Sender	String
Sensitivity	String
SMTPPort	Number
SMTPServer	String
SMTPSSLClientCertAlias	String
SMTPSSLPort	Number
SMTPUserName	String
SMTPUserPassword	String
SMTPUseSSL	Boolean
StatusNotifyOptions	String
StatusNotifyReturn	String
Subject	String
Text	String
TimeToWaitForResult	Number
UsedDefaultConfig	Boolean RO
UsedPrimaryServer	Boolean RO
ValidSentAddresses	String RO
ValidUnsentAddresses	String RO

SMTPSession Methods

Method	*Returns*
Send*(&Email)*	Constant
SendAll*(&Emails)*	Array of numbers
SetSMTPParam*(ParamName,ParamValue)*	None

SMTPSession Properties

Property	***Returns***
BackupPort	Number
BackupServer	String
BackupSSLClientCertAlias	String
BackupSSLPort	Number
BackupUserName	String
BackupUserPassword	String
BackupUseSSL	Boolean
IsAuthenticationReqd	Boolean
Port	Number
Server	String
SSLClientCertAlias	String
SSLPort	Number
UsedDefaultConfig	Boolean ^{RO}
UsedPrimaryServer	Boolean ^{RO}
UserName	String
UserPassword	String
UseSSL	Boolean

MCF IM Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the MCF IM classes.

MCFIMInfo Functions

Function	***Returns***
CreateMCFIMInfo*(UserID,Network)*	MCFIMInfo object

MCFIMInfo Methods

Method	***Returns***
AddUser*(User)*	Boolean
CheckAll()	Boolean
GetAdditionalUserInfo()	String
GetErrorImageName()	String
GetOffLineImageName()	String
GetOnlineImageName()	String
GetUnknownImageName()	String
GetLaunchURL*(User)*	String
GetStatus*(User)*	Integer
RemoveUser*(User)*	Boolean

MCFIMSingleButton Methods

Method	***Returns***
generateHTML*(&RS,slide_dir,X_pos,Y_pos,width, height,seq_num)*	String
generateJavaScript()	String
insertXMPPServerUserData*(PS_user_ID,protocol, XMPP_domain,IM_user_ID, IM_pwd)*	Boolean
MCFIMSingleButton()	MCFIMSingleButton object

Method	*Returns*
updateXMPPServerUserData*(PS_user_ID,protocol, XMPP_domain,IM_user_ID, IM_pwd)*	Boolean

MCFIMSingleButton Properties

Method	*Returns*
hideDelay	Integer

Message Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the message classes.

Message Functions

Function	*Returns*
AddSystemPauseTimes*(StartDay,StartTime,EndDay, EndTime*)	Boolean
CreateMessage(**OPERATION.***messagename* [, *Message_Type*])	Message object
DeleteSystemPauseTimes*(StartDay,StartTime,EndDay, EndTime*)	Boolean
GetNRXmlDoc(*NRID,EntityName)*	XmlDoc object
IBPurgeDomainStatus()	Boolean
IBPurgeNodeStatus()	Boolean
NodeDelete*(nodeName)*	Boolean
NodeRename*(oldNodeName,newNodeName)*	Boolean

Function	***Returns***
NodeSaveAs*(oldNodeName,newNodeName)*	Boolean
PingNode*(MsgNodeName)*	Array of number
PSIGWServiceRequest*(&Req_XmlDoc, &Resp_XmlDoc)*	Integer
ReValidateNRXmlDoc*(NRID,EntityName)*	Boolean

Message Methods

Method	***Returns***
Clone()	Message object
CopyPartRowset*(PartIndex,&Rowset)*	None.
CopyRowset*(source_rowset* [, *record_list*])	None
CopyRowsetDelta*(source_rowset* [, *record_list*])	None
CopyRowsetDeltaOriginal(*source_rowset*, [, *record_list*])	None
CreateNextSegment()	None
DeleteSegment*(Segment)*	None
ExecuteEdits([*editlevels*])	None
GenXMLPartString*(PartIndex)*	String
GenXMLString()	String
GetContentString([*SegmentIndex*])	String
GetDocument([*segmented_messsage*])	Document object

Method	***Returns***
GetPartAliasName*(PartIndex)*	String
GetPartName*(PartIndex)*	String
GetPartRowset*(PartIndex)*	Rowset object
GetPartVersion*(PartIndex)*	String
GetPartXmlDoc*(PartIndex)*	XmlDoc object
GetRowset([*version*])	Rowset object
GetSegment*(Segment)*	None
GetXmlDoc()	XmlDoc object
LoadXMLPartString*(PartIndex,XmlString)*	None
LoadXMLString*(XMLString)*	None
SegmentRestart*(TransactionID,Segment_index, segmentByDB)*	None
SetEditTable*(%PromptTable,***RECORD.***recname)*	None
SetXmlDoc*(&XmlDoc)*	None
UpdateSegment()	None

Message Properties

Property	***Returns***
ActionName	String RO
AliasName	String RO

Property	***Returns***
CurrentSegment	Number ^{RO}
DoNotPubToNodeName	String
IBInfo	IBInfo object ^{RO}
IsBulkLoadTruncation	Boolean ^{RO}
IsDelta	Boolean ^{RO}
IsEditError	Boolean ^{RO}
IsEmpty	Boolean ^{RO}
IsLocal	Boolean ^{RO}
IsOperationActive	Boolean ^{RO}
IsParts	Boolean^{RO}
IsPartsStructured	Boolean^{RO}
IsRequest	Boolean^{RO}
IsSourceNodeExternal	Boolean^{RO}
IsStructure	Boolean ^{RO}
Name	String ^{RO}
NRId	String ^{RO}
OperationName	String^{RO}
OperationVersion	String^{RO}

Property	***Returns***
ParentTransactionId	String^{RO}
PartCount	Number^{RO}
PubNodeName	String^{RO}
QueueName	String^{RO}
QueueSeqId	Number^{RO}
ResponseStatus	Number ^{RO}
SegmentCount	Number ^{RO}
SegmentsByDatabase	Boolean
Size	Number ^{RO}
TransactionId	String^{RO}
Version	String^{RO}

IntBroker System Variable

This section lists the system variables, as well as returns (if applicable) for the IntBroker class.

System Variable	***Returns***
%IntBroker	IntBroker object

IntBroker Methods

Method	***Returns***
Cancel*(TransactionId,QueueName,DataType, SegmentIndex | TransactionIdArray,QueueNameArray, DataTypeArray,SegmentIndexArray)*	Boolean
ConnectorRequest*(&Message)*	Message object

Method	***Returns***
ConnectorRequestUrl*(URL)*	String
DeleteOrphanedSegments*(TransactionId)*	Boolean
GetArchData*(TransactionId,SegmentIndex)*	String
GetArchIBInfoData*(TransactionId,ParentTransactionId)*	String
GetIBInfoData*(TransactionId,DataType)*	String
GetIBTransactionIDforAE*(OperID,RunCtlID)*	String
GetMessage([*TransactionId,DataType*])	Message object
GetMessageErrors*(TransactionId)*	Array of string
GetMsgSchema*(MsgName,MsgVersion)*	String
GetSyncIBInfoData*(TransactionId,%LogType* [, *Archive*])	String
GetSyncLogData*(TransactionId,%LogType* [, *Archive*])	String
InBoundPublish(*&Message)*	Boolean
IsOperationActive*(OperationName* [,*OperationVersion*])	Boolean
MsgSchemaExists*(MsgName,MsgVersion)*	Boolean
Publish*(&Message* [, *&ArrayofNodeNames*] [,*IsEnqueued*])	None.
Resubmit({*TransactionId,QueueName,DataType, SegmentIndex*} | {*TransactionIdArray,QueueNameArray, DataTypeArray,SegmentIndexArray*})	Boolean
SetMessageError*(TransactionID,MsgSet,MsgNumber, Error_Location, ParamListCounter,* [*Param*] ...)	Boolean
SetStatus(*&Message,Status)*	None
SwitchAsyncEventUserContext*(UserID,LanguageCode)*	Boolean
SyncRequest([*&MsgArray,&NodeNames*])	Array of messages
Update*(&Message)*	Boolean
UpdateXmlDoc*(&XmlDoc,TransactionId,DataType)*	Boolean

IBInfo Methods

Method	***Returns***
AddAEAttribute*(Name,value*	Boolean
AddAttachment*(Path)*	String
AddAttribute*(name,value)*	Boolean
AddContainerAttribute*(Name,Value)*	Boolean
ClearAEAttributes()	None
ClearAttachments()	None
ClearAttributes()	None
ClearContainerAttributes()	None
DeleteAEAttribute*(Name)*	Boolean
DeleteAttachment*(Index* | *Content_Id)*	Boolean
DeleteAttribute*(name)*	Boolean
DeleteContainerAttribute*(Name)*	Boolean
GetAEAttributeName*(nIndex)*	String
GetAEAttributeValue*(nIndex)*	String
GetAttachmentContentID*(Index)*	String
GetAttachmentProperty*(Content_ID,Property_Type)*	String
GetAttributeName*(nIndex)*	String
GetAttributeValue*(nIndex)*	String
GetContainerAttributeName*(nIndex)*	String
GetContainerAttributeValue*(nIndex)*	String
GetNumberofAEAttributes()	Integer
GetNumberOfAttributes()	Integer
GetNumberOfContainerAttributes()	Integer

Method	***Returns***
GetTransactionIDforAE()	Number
InsertAEResponseAttributes()	Boolean
LoadConnectorProp*(ConnectorName)*	Boolean
LoadConnectorPropFromNode*(NodeName)*	Boolean
LoadConnectorPropFromRouting*(RoutingDefnName)*	Boolean
SetAttachmentProperty*(Content_ID,Property_Type, Value)*	Boolean

IBInfo Properties

Property	***Returns***
AppServerDomain	String
CompressionOverride	String
ConnectorOverride	Boolean
ConversationId	String
DeliveryMode	Number
DestinationNode	String, RO
ExternalMessageID	Number RO
ExternalOperationName	String
ExternalUserName	String
ExternalUserPassword	String
FinalDestinationNode	String, RO
FuturePublicationDateTime	Datetime
HTTPSessionId	String
IBConnectorInfo	IBConnectorInfo collection object, RO
InReplyToID	String
MessageName	String, RO

Property	***Returns***
MessageQueue	String^{RO}
MessageType	String, ^{RO}
MessageVersion	Number, ^{RO}
NodeDN	String, ^{RO}
NonRepudiationID	String, ^{RO}
NumberOfAttachments	String, ^{RO}
OperationType	String, ^{RO}
OperationVersion	String, ^{RO}
OrigNode	String, ^{RO}
OrigProcess	String, ^{RO}
OrigTimeStamp	String, ^{RO}
OrigUser	String, ^{RO}
RequestingNodeName	String, ^{RO}
RequestingNodeDescription	String, ^{RO}
ResponseAsAttachment	Boolean
SegmentsUnOrder	Boolean
SourceNode	String, ^{RO}
SyncServiceTimeout	Number
TransactionID	String, ^{RO}
UserIed	String, ^{RO}
VisitedNodes	Array of string, ^{RO}
WSA_Action	String
WSA_FaultTo	String
WSA_MessageID	String
WSA_ReplyTo	String

Property	*Returns*
WSA_To	String

IBConnectorInfo Collection Methods

Method	*Returns*
AddConnectorProperties*(Name,Value,Type)*	Boolean
AddQueryStringArg*(Name,Value)*	Boolean
ClearConnectorProperties()	None
ClearQueryStringArgs()	None
DeleteConnectorProperties*(Name)*	Boolean
DeleteQueryStringArg*(Name)*	Boolean
GetConnectorPropertiesName*(Index)*	String
GetConnectorPropertiesType*(Index)*	String
GetConnectorPropertiesValue*(Index)*	String
GetNumberOf ConnectorProperties()	Number
GetNumberOfQueryStringArgs()	Number
GetQueryStringArgName*(Index)*	String
GetQueryStringArgValue*(Index)*	String

IBConnectorInfo Collection Properties

Property	*Returns*
ConnectorClassName	String
ConnectorName	String
Cookies	String
PathInfo	String
RemoteFrameworkURL	String

Notification Classes

This section lists the constructors, methods, and properties, as well as returns (if applicable) for the Notification, NotificationAddress, NotificationTemplate and WorklistEntry classes.

Constructors

Constructors	***Returns***
Notification*(NotifyFrom,dttmCreated,language_cd)*	Notification object
NotificationAddress*(Oprid,Description,Language, EmailId, Channel)*	NotificationAddress object
NotificationTemplate*(ComponentId,Market,TemplateId, TemplateType*)	NotificationTemplate object
OnAckForMarkedWorkedResp()	OnAckForMarkedWorkedResp object
WLEntryMarkedWorkedResp()	WLEntryMarkedWorkedResp object
WorkList()	Worklist object
WorklistEntry()	WorklistEntry object
WSWorklistEntry()	WSWorklistEntry object
WSWorklistEntryStatus()	WSWorklistEntryStatus object

Notification Class Method

Method	***Returns***
Send()	None

Notification Class Properties

Property	***Returns***
ContentType	String
dttmCreated	Datetime
EmailReplyTo	String
FileNames	Array of String
FileTitles	Array of String
language_cd	String
Message	String
NotifyBCC	Array of NotificationAddress objects
NotifyCC	Array of NotificationAddress objects
NotifyFrom	String
NotifyGuid	GUID
NotifyTo	Array of NotificationAddress objects
Rte	Boolean
SourceComponent	String
SourceMarket	String
SourceMenu	String
Subject	String
Template	String

NotificationAddress Properties

Property	***Returns***
Channel	String
Description	String
EmailId	String
Language	String
Oprid	String

NotificationTemplate Methods

Method	***Returns***
GetAndExpandTemplate*(Language,Vars)*	Boolean
SetupCompVarsAndRcpts*(&Rowset_Context)*	String
SetupGenericVars*(&AryValues)*	String

NotificationTemplate Properties

Property	***Returns***
ComponentId	String
Instruction	String
Language	String
Market	String
Priority	String

Property	***Returns***
Responses	String
Subject	String
TemplateId	String
TemplateType	String
Text	String

Worklist Method

Method	***Returns***
Reassign*(FromOperid,ToOperid)*	Boolean

WorklistEntry Methods

Method	***Returns***
Create()	Number
GetResponseStatus()	Number
Reassign*(Operid)*	Boolean
Save()	Number
SaveWithCustomData*(&Message,&FieldNameArray, &FieldValueArray)*	Message object`
SelectByKey()	Boolean
SelectByMessageId()	Boolean
Update()	Number

WorklistEntry Properties

Property	***Returns***
actiondttm	Datetime
busactivity	String
buseventname	String
busprocname	String
commentshort	String
do_replicate_flag	String
instanceid	Number
instselectddttm	Datetime
inststatus	String
insttimeoutddttm	Datetime
instworkeddttm	Datetime
IsCreatedViaWebService	Boolean ^{RO}
oprid	String
originatorid	String ^{RO}
prevoprid	String
requestmessageid	String
ResponseStatus	String ^{RO}

Property	*Returns*
syncid	Number
timedout	String
transactionid	Number
url	String
wl_priority	String
wldaystoselect	Number
wldaystowork	Number
worklistname	String

WSWorklistEntry Methods

Method	*Returns*
OnError*(&Message)*	Depends on implementation
OnNotify*(&Message)*	None

WSWorklistEntry Properties

Property	*Returns*
InstanceId	String RO
TransactionId	String RO

Optimization PeopleCode Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the optimization PeopleCode classes.

Important! Optimization PeopleCode classes are not supported on IBM z/OS and Linux for IBM System z.

Functions

Function	*Returns*
CreateOptEngine(*probinst*, {**%Synch** | **%ASynch**} [, *detailedstatus*] [,*processinstance*])	OptEngine object
CreateOptInterface()	OptInterface object
DeleteOptProbInst(*probinst*[, *detailedstatus*])	Constant
GetOptEngine(*probinst*[, *detailedstatus*])	OptEngine object
GetOptProbInstList(*ProblemType* , *OutputErrorCode* [, *Prefix*][, *detailedstatus*])	Array of string
InsertOptProbInst(*probinst*, *ProblemType*[, *detailedstatus*] [, *Description*])	Constant
IsValidOptProbInst(*probinst*[, *status*])	Constant

OptEngine Methods

Method	*Returns*
CheckOptEngineStatus()	Number
FillRowset(*PARAM_NAME*, *&Rowset* [, *functionstatus*])	Constant
GetDate(*PARAM_NAME*[,*status*])	Date
GetDateArray(*PARAM_NAME)*	Array of Date
GetDateTime(*PARAM_NAME)*	DateTime
GetDateTimeArray(*PARAM_NAME)*	Array of Datetime
GetNumber(*PARAM_NAME)*	Number

Method	***Returns***
GetNumberArray*(PARAM_NAME)*	Array of number
GetString*(PARAM_NAME)*	String
GetStringArray*(PARAM_NAME)*	Array of string
GetTime*(PARAM_NAME)*	Time
GetTimeArray*(PARAM_NAME)*	Array of Time
GetTraceLevel*(component)*	Constant
RunAsynch*(TRANSACTION, PARM_PAIRS)*	Constant
RunSynch*(TRANSACTION, PARM_PAIRS)*	Constant
SetTraceLevel*(Component, Severity)*	Constant
ShutDown()	Constant

OptEngine Properties

Property	***Returns***
DetailMsgs	Array of String ^{RO}
DetailStatus	Constant

OptBase Methods

Method	***Returns***
GetParmDate*(ParmName,&ParmValue)*	Boolean
GetParmDateArray*(ParmName,&ParmValue)*	Boolean
GetParmDateTime*(ParmName,&ParmValue)*	Boolean

Method	***Returns***
GetParmDateTimeArray*(ParmName,&ParmValue)*	Boolean
GetParmNumber*(ParmName,&ParmValue)*	Boolean
GetParmNumberArray*(ParmName,&ParmValue)*	Boolean
GetParmInt`*(ParmName,&ParmValue)*	Boolean
GetParmIntArray*(ParmName,&ParmValue)*	Boolean
GetParmString*(ParmName,&ParmValue)*	Boolean
GetParmStringArray*(ParmName,&ParmValue)*	Boolean
GetParmTime*(ParmName,&ParmValue)*	Boolean
GetParmTimeArray*(ParmName,&ParmValue)*	Boolean
Init()	Boolean
OptDeleteCallback*(&record)*	Boolean
OptInsertCallback*(&Record)*	Boolean
OptPostUpdateCallback*(&OldRecord,&NewRecord)*	Boolean
OptPreUpdateCallback*(&OldRecord,&NewRecord)*	Boolean
OptRefreshCallback()	Boolean
SetOutputParmDate*(ParmName,&ParmValue)*	Boolean
SetOutputParmDateArray*(ParmName,&ParmValue)*	Boolean
SetOutputParmDateTime*(ParmName,&ParmValue)*	Boolean
SetOutputParmDateTimeArray*(ParmName,&ParmValue)*	Boolean
SetOutputParmNumber*(ParmName,&ParmValue)*	Boolean
SetOutputParmNumberArray*(ParmName,&ParmValue)*	Boolean
SetOutputParmInt*(ParmName,&ParmValue)*	Boolean
SetOutputParmIntArray*(ParmName,&ParmValue)*	Boolean
SetOutputParmString*(ParmName,&ParmValue)*	Boolean
SetOutputParmStringArray*(ParmName,&ParmValue)*	Boolean

Method	***Returns***
SetOutputParmTime*(ParmName,&ParmValue)*	Boolean
SetOutputTimeArray*(ParmName,&ParmValue)*	Boolean

OptInterface Methods

Method	***Returns***
ActivateModel*(ModelID,SolverSettingID*	Constant
ActivateObjective*(Model_Name,Objective_Name)*	Constant
DeactivateModel*(ModelId)*	Constant
DumpMsgToLog*(LogSeverity,Message)*	None
FindRowNum*(&Record*[, *StartRow*[, *EndRow* [, *field_list*]]]	Number
GetSolution*(ModelId,varArrayId,SkipZero* [, *KeyFieldNames, KeyFieldValues*[, *Sollution*]]	Constant
GetSolutionDetail(*ModelId,SolutionType,Name,&Solution*)	Constant
IsModelActive(*ModelId)*	Boolean
RestoreBounds*(ModelId*[, *varArrayId*])	Constant
SetVariableBoungs*(ModelId,varArrayId,BoundType, LowerBound, UpperBound,&KeyRecord*[, *ChangeModelBounds*])	Constant
SetVariableType(*ModelId,varArrayId,varType)*	Constant
Solve(*ModelId,SolutionType*[, *objValue*[, *name-value_pairs*]])	Constant

Page

This section lists the functions, methods, and properties, as well as returns (if applicable) for the page class.

Function

Function	***Returns***
GetPage(**PAGE.***pagename*)	Page object

Properties

Property	***Returns***
CopyURLLink	Boolean
CustomizePageLink	Boolean
DisplayOnly	Boolean RO
HelpLink	Boolean
Name	String RO
NewWindowLink	Boolean
Visible	Boolean

PortalRegistry

See Appendix A, "Quick Reference for PeopleCode Classes," PortalRegistry Classes Methods and Properties, page 2809.

PostReport

This section lists the functions, methods, and properties, as well as returns (if applicable) for the PostReport class.

Function

Function	***Returns***
SetPostReport()	PostReport object

Methods

Method	Returns
AddDistributionOption*(DistIdType, DistId)*	Number
Put()	

Properties

Property	Returns
ExpirationDate	Date
OutDestFormat	String
ProcessInstance	Number
ProcessName	String
ProcessType	String
ReportDesc	String
ReportFolder	String
ReportId	Number
ServerName	String

Process Request Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the ProcessRequest class and the related PrcsApi class.

ProcessRequest Functions

Function	***Returns***
CreateProcessRequest([*ProcessType*,*ProcessName*])	ProcessRequest object
GetNextProcessInstance([*Commit*])	Number
SetupScheduleDefnItem*(ScheduleName, JobName)*	ProcessRequest object

ProcessRequest Methods

Method	***Returns***
AddDistributionOption*(DistIdType*,*DistId* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])	Number
AddNotifyInfo*(field_name*,*field_value* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])	None
PrintJobHTMLRpt()	String
PrintJobRqstRpt(*ProcessInstance*,*ItemInstance* [, *PrintJobTree*] [, *PrintDistList*] [, *PrintNotifyList*] [, *PrintSystemMessage*] [, *PrintApplicationMessage*] [, *PrintParamList*])	String
PrintSchdlHTMLRpt([*PrintJobTree*] [, *PrintDistList*] [, *PrintNotifyList*] [, *PrintMessageList*] [, *PrintParamList*])	String
RunJobSetNow()	None
Schedule()	None
SetEmailOption*(EmailSubject*,*EmailText*,*EmailAddress*, *EmailWebReport*, *EmailAttachLog* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])	None
SetItemFolder(*PortalFolder* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])	None

Method	***Returns***
SetNotifyAppMethod*(app_class_name, app_class_method* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])	None
SetNotifyService(*srvc_op_name*[, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])	None
SetOutputOption*(OutputType,OutputFormat, OutputDest* [, *JobName*][, *PrcsItemLevel*] [, *JobSeqNo*])	None
UpdateRunStatus()	None

ProcessRequest Properties

Property	***Returns***
EmailAttachLog	Boolean
EmailSubject	String
EmailText	String
EmailWebReport	Boolean
FileName	String
JobName	String
LanguageCd	String
NotifyTextMsgNum	Number
NotifyTextMsgSet	Number
OutDest	String
OutDestFormat	String

Property	*Returns*
OutDestType	String
PortalFolder	String
ProcessInstance	Number
ProcessName	String
ProcessType	String
RunControlID	String
RunDateTime	DateTime
RunLocation	String
RunRecurrence	String
RunStatus	Number
Status	Number [RO]
TimeZone	String

PrcsApi Constructor

Constructor	*Returns*
PrcsApi()	PrcsApi object

PrcsApi Methods

Method	*Returns*
getAllFileNames(*PrcsInstance)*	Array of string
notifyToWindows*(PrcsInstance,Message)*	Number

Query

See Appendix A, "Quick Reference for PeopleCode Classes," Query Classes Methods and Properties, page 2837.

Record

This section lists the functions, methods, and properties, as well as returns (if applicable) for the record class.

Functions

Function	***Returns***
CreateRecord**(RECORD.***recname*)	Record object
GetRecord([**RECORD.***recname*])	Record object

Methods

Method	***Returns***
CompareFields*(RecordObject)*	Boolean
CopyChangedFieldsTo*(RecordObj)*	None
CopyFieldsTo*(RecordObject)*	None
DBPatternMatch*(Value,Pattern,CaseSensitive)*	Boolean
Delete()	Boolean
ExecuteEdits([*EditLevel*])	None
GetField*(n/***FIELD.***fieldname)*[D]	Field Object
Insert()	Boolean
Save([*CopyToOriginal*])	Boolean

Method	***Returns***
SearchClear()	None
SelectByKey([*UseChangedBuffers*])	Boolean
SelectByKeyEffDt(*Date* [, *UseChangedBuffers*])	Boolean
SetDefault()	None
SetEditTable(*%PromptField,***RECORD.***recname)*	None
Update([*KeyRecord*])	Boolean

Properties

Property	***Returns***
FieldCount	Number RO
fieldname	Field Object RO
IsChanged	Boolean RO
IsDeleted	Boolean RO
IsEditError	Boolean RO
Name	String RO
ParentRow	Row Object RO
RelLangRecName	String RO

Row

This section lists the functions, methods, and properties, as well as returns (if applicable) for the row class.

Function

Function	***Returns***
GetRow()	Row object

Methods

Method	***Returns***
CopyTo*(rowobject)*	Row Object
GetNextEffRow()	Row Object
GetPriorEffRow()	Row Object
GetRecord({*n* | **RECORD.***recname*})^{D}	Record Object
GetRowSet({*n* | **SCROLL.***scrollname*})	Rowset Object
scrollname(n)	Row Object

Properties

Property	***Returns***
ChildCount	Number ^{RO}
DeleteEnabled	Boolean
IsChanged	Boolean ^{RO}
IsDeleted	Boolean ^{RO}
IsEditError	Boolean ^{RO}
IsNew	Boolean ^{RO}

Property	***Returns***
ParentRowSet	Rowset object RO
recname	Record object RO
RecordCount	Number RO
RowNumber	Number RO
Selected	Boolean
Style	String
Visible	Boolean

Rowset

This section lists the functions, methods, and properties, as well as returns (if applicable) for the rowset class.

Functions

Function	***Returns***
CreateRowset({**RECORD.***recname* | *&Rowset*} [, { **FIELD.***fieldname*,**RECORD.***recname* | *&Rowset*}] . . .)	Rowset object
GetLevel0()	Rowset object
GetRowset([SCROLL.*scrollname*])	Rowset object

Methods

Method	***Returns***
ClearDeletesChanges()	None

Method	***Returns***
CopyTo*(&DestRowset* [, **RECORD.***SourceRecname*, **RECORD.***DestRecname*]. . .)	None
DeleteRow*(n)*	Boolean
Fill([*wherestring* [, *bindvalue*] . . .])	Number
FillAppend([*wherestring* [, *bindvalue*] . . .])	Number
Flush()	None
FlushRow*(n)*	None
GetCurrEffRow()	Row object
GetFirstUserSortedRow*(GridName)*	Row object
GetLastUserSortedRow*(GridName* [, *visible*])	Row object
GetNewEffRow()	Row object
GetRow*(n)*[D]	Row object
HideAllRows()	None
InsertRow*(n)*	Boolean
IsUserSorted*(GridName)*	Boolean
MapBufRowToUserSortRow*(GridName,number)*	Number
MapUserSortRowToBufRow*(GridName,number)*	Number
Refresh()	None
Select([*parmlist*], **RECORD.***selrecord* [, *wherestr*, *bindvars*])	Number

Method	***Returns***
SelectNew([*parmlist*], **RECORD.***selrecord* [, *wherestr, bindvars*])	Number
SetDefault*(recname.fieldname)*	None
ShowAllRows()	None
Sort([*paramlist,*] *sort_fields)*	None

Properties

Property	***Returns***
ActiveRowCount	Number ^{RO}
ChangeOnInit	Boolean
DataAreaCollapsed	Boolean
DBRecordName	String ^{RO}
DeleteEnabled	Boolean
EffDt	Date ^{RO}
EffSeq	Number ^{RO}
InsertEnabled	Boolean
IsEditError	Boolean ^{RO}
Level	Number ^{RO}
Name	String ^{RO}
ParentRow	Row Object ^{RO}

Property	*Returns*
ParentRowSet	Rowset Object ^{RO}
RowCount	Number ^{RO}
SetComponentChanged	Boolean
TopRowNumber	Number ^{RO}

RowsetCache

This section lists the functions, methods, and properties, as well as the returns (if applicable) for the RowsetCache class.

Functions

Function	*Returns*
CreateRowsetCache*(&Rowset,* [**Rowset.**]*Name, Description)*	RowsetCache object
GetRowsetCache(([**Rowset.**]*name)*	RowsetCache object

Methods

Method	*Returns*
Delete()	Boolean
Get()	Rowset object
Save([[**Rowset.**]*name*] [, *Description*] [, *Lang*])	Boolean

Properties

Property	*Returns*
Content	Rowset object
Description	String

Search

See Appendix A, "Quick Reference for PeopleCode Classes," Verity Search Classes Methods and Properties, page 2800.

Session

See Appendix A, "Quick Reference for PeopleCode Classes," Session Classes, page 2787.

SOAPDoc

This section lists the functions, methods, and properties, as well as returns (if applicable) for the SOAPDoc class.

Function

Function	*Returns*
CreateSOAPDoc()	SOAPDoc object

Methods

Method	*Returns*
AddBody()	None
AddEnvelope*(SOAP_Schema)*	None
AddFault*(Fault_Code,Fault_String)*	None
AddHeader()	None
AddMethod*(MethodName,IsRequest)*	None
AddParm*(ParmName,ParmValue)*	None
GetParmName*(Index)*	String
GetParmValue*(Index)*	String

Method	***Returns***
ValidateSOAPDoc([1])	String

Properties

Property	***Returns***
BodyNode	XmlNode object RO
EnvelopeNode	XmlNode object RO
FaultCode	Number RO
FaultCodeS	String RO
FaultString	String RO
HeaderNode	XmlNode object RO
MethodName	String RO
MethodNode	XmlNode object RO
ParmCount	Number RO
XmlDoc	XmlDoc object

SQL

This section lists the functions, methods, and properties, as well as returns (if applicable) for the SQL class.

Functions

Function	***Returns***
CreateSQL([{*sqlstring*| **SQL.***SqlName*} [, *paramlist*]])	None

Function	***Returns***
DeleteSQL([**SQL.**]*sqlname* [, *dbtype* [, *effdt*]]])	Boolean
FetchSQL([**SQL.**]*sqlname* [, *dbtype* [, *effdt*]])	String
GetSQL(**SQL.***sqlname* [, *paramlist*])	SQL object
StoreSQL(*sqlstring*, [**SQL.**]*sqlname* [, *dbtype* [, *effdt* [, *ownerId* [, *Description*]]]])	None

Methods

Method	***Returns***
Close()	Boolean
Execute*(paramlist)*	Boolean
Fetch*(paramlist)*	Boolean
Open(*sql* [, *paramlist*])	None

Properties

Property	***Returns***
BulkMode	Boolean
IsOpen	Boolean ^{RO}
LTrim	Boolean
ReuseCursor	Boolean
RowsAffected	Number ^{RO}
Status	Number ^{RO}

Property	Returns
TraceName	String
Value	String RO

TransformData

This section lists the variables, properties, and returns (if applicable) for the TransformData class.

System Variable

System Variable	Returns
%TransformData	TransformData object

Properties

Property	Returns
DestMsgName	String RO
DestMsgVersion	String RO
DestNode	String RO
RejectTransform	Number
RoutingDefnName	String RO
SourceMsgName	String RO
SourceMsgVersion	String RO
SourceNode	String RO
Status	Number

Property	***Returns***
XmlDoc	XmlDoc object

Tree

See Appendix A, "Quick Reference for PeopleCode Classes," Tree Classes Methods and Properties, page 2857.

Universal Queue

This section lists the built-in functions, constructors, methods, and properties as well as returns (if applicable) for the universal queue classes (MultiChannel Framework).

Universal Queue Built-in Functions

Function	***Returns***
DeQueue*(physical queue ID,task type,task number, agent ID)*	Number
EnQueue*(logical queue ,task type,Relative URL, Language_Code* [,*subject*][, *agent ID*][, *overflow timeout*][, *escalation timeout*][, *cost*][, *priority*][, *skill level*])	Number
Forward*(from physical queue ID,from agent ID,task number,task type,to logical queue ID*[, *to agent ID*])	Number
InitChat(*logical queue ID,application data URL, customer username,* [*chat_subject*][, *chat_question*][, *wizard_URL*][, *priority*][, *skill_level*] [, *cost*])	Number
MCFBroadcast*(ClusterID,QueueID,ChannelID, AgentState,AgentPresence,Message, MessageSetNumber,MessageNumber,DefaultMessage, SecurityLevel,ImportanceLevel,SenderId, NameValueString)*	None.
NotifyQ*(logical queue ID,task type)*	Number

Universal Queue Constructors

Constructor	***Returns***
Agent*(AgentID)*	Agent object
AgentPhysQueueProps()	Agent physical queue properties object
AgentPhysQueueTasks*(PhysicalQueueID,AgentID)*	Agent physical queue tasks object
LogicalQueue*(LogicalQueueID)*	Logical queue object
MCFFactory([*MAX_TASKLIST_ITEMS*])	MCFFactory object
PhysicalQueue*(PhysicalQueueID)*	Physical queue object
Task*(TaskNumber)*	Task object
TaskList*(Status,TaskType,* [*PhysicalQueueID*[, *AgentID*]])	Task list object
Util()	Util object

Agent Methods

Method	***Returns***
Delete()	Number
Refresh()	None
RefreshQTaskList*(PhysQ)*	None

Agent Properties

Property	***Returns***
AgentID	String ^{RO}
AgentProps	AgentPhysQueueProps object ^{RO}
AgentTasks	AgentPhysQueueTask object ^{RO}
Buddy	Array of string ^{RO}
Language	Array of string ^{RO}
Name	String ^{RO}
Nickname	String ^{RO}

Property	Returns
PhysicalQueueID	Array of string RO
TotalPhysicalQueues	Number RO

AgentPhysQueueProps Properties

Property	Returns
AgentID	String RO
PhysicalQueueID	String RO
SkillLevel	String RO
WorkLoad	Number RO

AgentPhysQueueTasks Method

Method	Returns
Refresh*(TaskList)*	None.

AgentPhysQueueTasks Properties

Property	Returns
AcceptedTaskList	Task list object RO
AssignedTaskList	Task list object RO
PhysicalQueueID	String RO

LogicalQueue Properties

Property	Returns
LogicalQueueID	String RO
PhysicalQueueID	Array of PhysicalQueue objects RO

MCFFactory Property

Property	Returns
LogicalQueue	Array of LogicalQueue objects RO

PhysicalQueue Methods

Method	Returns
Refresh()	None
RefreshTaskList*(TaskList)*	None.

PhysicalQueue Properties

Property	Returns
AcceptedTaskList	Task list object RO
AssignedTaskList	Task list object RO
Agent	Array of agent objects RO
BrowserURL	String RO
EnqueuedTaskList	Task list object RO
EscalatedTaskList	Task list object RO
InternalURL	String RO
IsActive	Boolean RO
LogicalQueueID	String RO
OverflowedTaskList	Task list object RO
PhysicalQueueID	String RO
RENURLID	String RO
TotalAgents	Number RO

Task Methods

Method	Returns
Close*(Comment)*	None

Method	***Returns***
Enqueue*(LogicalQueueID,AgentID,Timeout, ResponseTime,Cost,Priority,MinSkill)*	None.
Refresh()	None
RefreshStatus()	None

Task Properties

Property	***Returns***
AgentID	String RO
ApplicationData	String RO
Comments	String
EnqueueTime	Datetime RO
EscalationTime	Datetime RO
Language	String RO
OriginalTime	Datetime RO
OverflowTime	Datetime RO
PhysicalQueueID	String RO
TiedAgentID	String RO

TaskList Method

Method	***Returns***
Refresh()	None

TaskList Properties

Property	***Returns***
AgentID	String RO
PhysicalQueueID	String RO
Task	Array of tasks RO

Property	***Returns***
TaskType	String RO
Total	Number RO

XmlDoc Classes

This section lists the functions, methods, and properties, as well as returns (if applicable) for the XmlDoc class and the XmlNode class.

XmlDoc Classes Functions

Function	***Returns***
CreateXmlDoc*(XmlString)*	XmlDoc object
GetNRXmlDoc*(NRID,NodeName)*	XmlDoc object
RevalidateNRXmlDoc*(NRID,EntityName*	Boolean
Transform*(Input, AE_Program_Name, Initial_Node_Name,Initial_Message__Name, Initial_Message_Version,Result_Node_Name, Result_Message_Name,Result_Message_Version)*	XmlDoc object
TransformEx*(XmlString,XsltString)*	String

XmlDoc Class Methods

Method	***Returns***
CopyRowset*(&InRowset*[, *MessageName*] [, *MessageVersion*])	Boolean
CopyToPSFTMessage*(targetDoc, srcPath, targetPath)*	Number
CopyToRowset*(&InRowset*[, *Message_Name*][, *Message_Version*])	Boolean
CreateDocumentElement*(TagName* [, *NamespaceURI*] [, *&DocType*])	XmlNode object

Method	***Returns***
CreateDocumentType(*Name,PublicID,SystemID)*	String
GenFormattedXmlString()	String
GenXmlFile*(path)*	Boolean
GenXmlString()	String
GetElementsByTagName*(TagName)*	Array of XmlNode objects
LoadIBContent*(Content*[, *RootTagName*])	Boolean
ParseXmlFromURL*(path)*	Boolean
ParseXmlString*(XmlString)*	Boolean

XmlDoc Class Properties

Property	***Returns***
DocumentElement	XmlNode object RO
IsNull	Boolean RO

XmlNode Class Methods

Method	***Returns***
AddAttribute*(Name,Value)*	None
AddAttributeNS*(NamespaceURI,AttributeName,Value)*	None
AddCDataSection*(Data)*	XmlNode object
AddComment*(Text)*	XmlNode object

Method	***Returns***
AddElement*(TagName)*	XmlNode object
AddElementNS*(NameSpaceURI,TagName)*	XmlNode object
AddEntityReference*(Name)*	XmlNode object
AddNode*(XmlNode)*	XmlNode object
AddProcessInstruction*(Target,Data)*	XmlNode object
AddText*(Data)*	XmlNode object
CopyNode*(Node)*	None
FindNode*(Path)*	XmlNode object
FindNodes*(Path)*	Array of XmlNode objects
GenXmlString()	String
GetAttributeName*(index)*	String
GetAttributeValue({*Name* | *Index*})	String
GetCDataValue()	String
GetCDataValues()	Array of string
GetChildNode*(index)*	XmlNode object
GetElement()	XmlNode object
GetElements()	Array of XmlNode objects
GetElementsByTagName*(TagName)*	Array of XmlNode objects

Method	***Returns***
GetElementsByTagNameNS*(NamespaceURI,TagName)*	Array of XmlNode objects
InsertCDataSection*(Data,Position)*	XmlNode object
InsertComment*(Data,Position)*	XmlNode object
InsertElement*(TagName,Position)*	XmlNode object
InsertElementNS*(NameSpaceURI,TagName,Position)*	XmlNode object
InsertEntityReference*(Name,Position)*	XmlNode object
InsertNode*(&NewNode,* {*&RefNode* | *Position*})	XmlNode object
InsertProcessInstruction*(Target,Data,Position)*	XmlNode object
InsertText*(Data,Position)*	XmlNode object
RemoveAllChildNode()	None
RemoveChildNode({*Position* | *Node*})	XmlNode object

XmlNode Class Properties

Property	***Returns***
AttributesCount	Number ^{RO}
ChildNodeCount	Number ^{RO}
Index	Number ^{RO}
IsNull	Boolean ^{RO}
LocalName	String ^{RO}

Property	*Returns*
NamespaceURI	String ^{RO}
NextSibling	XmlNode object ^{RO}
NodeName	String
NodePath	String ^{RO}
NodeType	String ^{RO}
NodeValue	String
ParentNode	XmlNode object ^{RO}
Prefix	String ^{RO}
PreviousSibling	XmlNode object ^{RO}

XML Publisher

This section lists the constructors, methods, and properties, as well as returns (if applicable) for the XML Publisher classes.

- Report manager definition classes
- Report manager search classes
- XML Publisher engine classes

Report Definition Manager Classes Constructors

Constructor	*Returns*
ReportDefn*(ReportId)*	ReportDefn object

ReportDefn Class Methods

Method	*Returns*
Close()	None

Method	***Returns***
DisplayOutput()	None
Get()	ReportDefn object
GetOutDestFormatString*(OutDestFormat)*	String
GetPSQueryPromptRecord()	Record object
PrintOutput*(DestPrinter)*	None
ProcessReport*(TemplateId,LanguageCD,AsOfDate, OutputFormat*)	None
Publish*(ServerName,ReportPath,FolderName, ProcessInstanceId*)	None
SetPSQueryPromptRecord(*&Record)*	None
SetRuntimeDataXMLFile*(FilePath)*	None
SetRuntimeProperties*(&NameArray,&ValueArray)*	None

ReportDefn Class Properties

Property	***Returns***
Description	String ^{RO}
DestinationPrinter	String
FolderName	String
OutDestination	String ^{WO}
OutDestinationType	String ^{RO}
ProcessInstance	String
Status	String^{RO}
UseBurstValueAsOutputFileName	Boolean

Report Manager Search Classes Constructors

Constructor	***Returns***
Report*(RptId,Prcsinstance,Contentid,Dbname,RptName, RptDescr,RptURL,RptCreateDttm,RptExpireDt, FldrName)*	Report object
ReportManager()	ReportManager object
SearchAttribute*(AttrName,AttrValue,CompareOperator)*	SearchAttribute object

Report Class Properties

Property	***Returns***
contentId	Number^{RO}
CreatedDateTime	DateTime^{RO}
DatabaseName	String^{RO}
Description	String^{RO}
ExpireDate	Date^{RO}
FileURL	String^{RO}
FolderName	String^{RO}
ProcessInstanceID	String^{RO}
ReportInstanceID	String^{RO}
ReportName	String^{RO}
ReportURL	String^{RO}

ReportManager Class Methods

Method	***Returns***
AddSearchFieldCriteria*(&SearchAttribute)*	Boolean
GetReportList()	Array of ReportDefn objects
SetBurstFieldCriteria*(BurstFld,BurstOp,BurstValue)*	Boolean
SetCaseSensitive*(IsCaseSensitive)*	None
SetDateCriteria*(createdDate,createdLastVal,createdUnit)*	None

Method	*Returns*
SetFolderCriteria*(FolderName)*	Boolean
SetProcessInstanceCriteria*(FromPID,ToPID)*	Boolean
SetReportIDCriteria*(ReportId)*	Boolean
SetUserIdCriteria*(UserId)*	Boolean

SearchAttribute Class Properties

Property	*Returns*
attrName	String^{RO}
attrValue	String^{RO}
compareOp	String^{RO}

XML Publisher Engine Classes Constructors

Constructor	*Returns*
PageNumber()	PageNumber object
PDFMerger()	PDFMerger object
Properties()	Properties object
Watermark()	Watermark object

PageNumber Class Properties

Property	*Returns*
BackgroundFile	String
FontName	String
FontSize	Number
PositionX	Number
PositionY	Number
StartFromPageNum	Number

Property	*Returns*
StartNum	Number

PDFMerger Class Method

Method	*Returns*
MergePDFs*(PDFFileArray,PDFOutputFile,Error)*	Boolean

PDFMerger Class Properties

Property	*Returns*
ConfProp	Properties object
Locale	Sting
PageNumber	PageNumber object
Watermark	Watermark object

Properties Class Methods

Method	*Returns*
GetProperty*(Name,OutValue)*	Boolean
SetProperty*(Name,Value)*	Boolean

Watermark Class Properties

Property	*Returns*
ImageFile	String
ImageFileLowerLeftX	Number
ImageFileLowerLeftY	Number
ImageFileUpperRightX	Number
ImageFileUpperRightY	Number
PageIndex	Number

Property	***Returns***
Text	String
TextAngle	Number
TextFontName	String
TextFontSize	Number
TextStartPosX	Number
TextStartPosY	Number

Session Classes

This section lists:

- Session classes methods and properties
- Component interface classes methods and properties
- Verity search classes methods and properties
- PortalRegistry classes methods and properties
- Query classes methods and properties
- Tree classes methods and properties

Session Classes Methods and Properties

A session object controls access to the PeopleSoft APIs as well as the environment in which they run. This section only lists the system variable, methods, and properties, as well as returns (if applicable) for the session class and the associated classes used for controlling the environment. The properties, methods, and so of the API objects (query classes, tree classes, and so on) are detailed in separate sections.

Session System Variable

System Variable	***Returns***
%Session	Session object

Session Methods

Method	***Returns***
AdvancedSearchQueries*(GetFavorites,QueryName, QueryNameOp,Descr, DescrOp,FolderName, FolderNameOp,RecordName,RecordNameOp, FieldName,FieldNameOp,TreeName,TreeNameOp, QueryType, OwnerType,CaseSensitive)*	Query collection
AdvancedSearchRecords*(RecordName,RecordNameOp, Descr,DescrOp, FieldName,FieldNameOp,TreeName, TreeNameOp,CaseSensitive*)	QueryDBRecord collection
FindCompIntfcs*(partial_name)*	Component Interface collection
FindPortalRegistries*(partial_name)*	PortalRegistry collection
FindQueryDBRecords()	QueryDBRecord collection
FindQueries()	Query collection
FindQueriesDateRange*(StartDateString,EndDateString)*	Query collection
GetActualRemoteNodes()	Remote node collection
GetCompIntfc([**CompIntfc.**]*name)*	Component Interface object
GetLocalNode()	Node object
GetNodes()	Node collection
GetPortalRegistry()	PortalRegistry object
GetQuery()	Query object
GetQuerySecurityProfile()	QuerySecurityProfile object
GetRemoteNodes()	Node collection
GetSearchIndexes()	SearchIndexes collection

Method	*Returns*
GetSearchQuery()	SearchQuery object
GetTree()	Tree object
GetTreeStructure()	Tree structure
SearchQueryDBRecord*(SearchType,Pattern, CaseSensitive)*	QueryDBRecord collection
SearchPrivateQueries*(QueryType, UserId, SearchType, Pattern,CaseSensitive)*	Query collection
SearchPublicQueries*(QueryType, SearchType,Pattern, CaseSensitive)*	Query collection

Session Properties

Property	*Returns*
ErrorPending	Boolean ^{RO}
PSMessages	PSMessage collection object
PSMessageMode	Number
RegionalSettings	Regional settings object
Repository	API Repository object
SuspendFormating	Boolean
TraceSettings	Trace settings object
WarningPending	Boolean ^{RO}

PSMessage Collection

Method	***Returns***
DeleteAll()	Boolean
DeleteItem*(number)*	Boolean
First()	PSMessage object
Item*(number)*	PSMessage object
Next()	PSMessage object

PSMessage Collection Property

Property	***Returns***
Count	Number [RO]

PSMessage Properties

Property	***Returns***
ExplainText	String [RO]
MessageNumber	Number [RO]
MessageSetNumber	Number [RO]
Source	String [RO]
Text	String [RO]

RegionalSettings Properties

Property	***Returns***
ClientTimeZone	String

Property	***Returns***
CurrencyFormat	Number
CurrencySymbol	String
DateFormat	Number
DateSeparator	String
DecimalSymbol	String
DigitGroupingSymbol	String
LanguageCD	String
UseLocalTime	Boolean
1159Separator	String
2359Separator	String

TraceSettings Properties

Property	***Returns***
API	Boolean
COBOLStmtTimings	Boolean
ConDisRollbackCommit	Boolean
DBSpecificCalls	Boolean
ManagerInfo	Boolean
NetworkServices	Boolean

Property	***Returns***
NonSSBs	Boolean
OutputUNICODE	Boolean
PCExtFcnCalls	Boolean
PCFcnReturnValues	Boolean
PCFetchedValues	Boolean
PCIntFcnCalls	Boolean
PCListProgram	Boolean
PCParameterValues	Boolean
PCProgramStatements	Boolean
PCStack	Boolean
PCStartOfPrograms	Boolean
PCTraceProgram	Boolean
PCVariableAssignments	Boolean
RowFetch	Boolean
SQLStatement	Boolean
SQLStatementVariables	Boolean
SSBs	Boolean
SybBindInfo	Boolean

Property	Returns
SybFetchInfo	Boolean
TraceFile	String

API Repository Classes Methods and Properties

This section lists the method, properties, as well as the returns (if applicable) for the API Repository classes.

Repository Properties

Property	Returns
Bindings	Bindings collection object RO
NameSpaces	Namespaces collection RO

Bindings Collection Property

Property	Returns
Count	Number RO

Bindings Collections Method

Method	Returns
Item*(number)*	Binding object

Bindings Property

Property	Returns
Name	String RO

Namespaces Collection Property

Property	Returns
Count	String RO

Namespaces Collection Methods

Method	***Returns***
Item*(Number)*	Namespaces object
ItemByName*(Name)*	Namespaces object

Namespaces Properties

Property	***Returns***
Classes	ClassInfo object ^{RO}
Name	String ^{RO}

ClassInfo Collection Property

Property	***Returns***
Count	Number ^{RO}

ClassInfo Collection Methods

Method	***Returns***
Item*(Number)*	ClassInfo object
ItemByName*(Name)*	ClassInfo object

ClassInfo Properties

Property	***Returns***
Documentation	String ^{RO}
Methods	MethodInfo collection object ^{RO}
Name	String ^{RO}
Properties	PropertyInfo collection object ^{RO}

MethodInfo Collection Property

Property	Returns
Count	String RO

MethodInfo Collection Methods

Method	Returns
Item*(Number)*	MethodInfo object
ItemByName*(Name)*	MethodInfo object

MethodInfo Properties

Property	Returns
Arguments	PropertyInfo collection object RO
Documentation	String RO
Name	String RO
Type	String RO

PropertyInfo Collection Properties

Property	Returns
Count	String RO

PropertyInfo Collection Methods

Method	Returns
Item*(Number)*	PropertyInfo object
ItemByName*(Name)*	PropertyInfo object

PropertyInfo Properties

Property	Returns
Documentation	String RO
Name	String RO

Property	***Returns***
Type	String RO
Usage	Number RO

Component Interface Class Methods and Properties

This section lists the methods and properties, as well as returns (if applicable) for the component interface classes.

Component Interface Collection Methods

Method	***Returns***
First()	Component Interface structure (no data)
Item*(number)*	Component Interface structure (no data)
Next()	Component Interface structure (no data)

Component Interface Collection Property

Property	***Retuns***
Count	Number RO

Component Interface Methods

Method	***Returns***
Cancel()	Boolean
CopyRowset*(&rowset* [,*IntialRow*] [, *record_list*])	None
CopyRowsetDelta*(&rowsetI,* [,*IntialRow*] [, *record_list*])	None
CopySetupRowset(*&rowset* [,*IntialRow*] [, *record_list*])	None

Method	***Returns***
CopySetupRowsetDelta*(&rowsetI*, [,*IntialRow*] [, *record_list*])	None
Create()	Component Interface with data
Find()	Component Interface collection
Get()	Component Interface with data
GetPropertyByName(*string)*	Depends on property
Save()	Boolean
SetPropertyByName*(string, value)*	None

Component Interface Properties

Property	***Returns***
ComponentName	String ^{RO}
CreateKeyInfoCollection	Component InterfacePropertyInfo collection object ^{RO}
FindKeyInfoCollection	Component InterfacePropertyInfo collection object ^{RO}
GetDummyRows	Boolean
GetHistoryItems	Boolean
GetKeyInfoCollection	Component InterfacePropertyInfo collection object ^{RO}
InteractiveMode	Boolean
PropertyInfoCollection	Component InterfacePropertyInfo collection object ^{RO}
StopOnFirstError	Boolean

Property	*Returns*
Every user-defined property in a component interface definition can be used like a property on the instantiated object.	Depends on property

CompIntfPropInfo Collection Methods

Method	*Returns*
First()	Component InterfacePropertyInfo object
Item*(number)*	Component InterfacePropertyInfo object
Next()	Component InterfacePropertyInfo object

CompIntfPropInfo Collection Property

Property	*Returns*
Count	Number RO

CompIntfPropInfo Properties

Property	*Returns*
Format	String RO
IsCollection	Boolean RO
IsReadOnly	Boolean RO
Key	Boolean RO
LabelLong	String RO
LabelShort	String RO

Property	***Returns***
Length	Number RO
Name	String RO
Prompt	Boolean RO
PropertyInfoCollection	Component InterfacePropertyInfo collection object
Required	Boolean RO
Type	String RO
Xlat	Boolean RO
YesNo	Boolean RO

Data Collection Methods

Method	***Returns***
CurrentItem()	Data collection item
DeleteItem*(number)*	None
GetEffectiveItem*(DateString,SeqNo)*	Data collection item
GetEffectiveItemNum*(DateString,SeqNo)*	Number
InsertItem*(number)*	Data collection item
Item*(number)*	Data collection item
ItemByKeys*(key_values)*	Data collection item

Data Collection Properties

Property	*Returns*
Count	Number ^{RO}
CurrentItemNumber	Number ^{RO}

Verity Search Classes Methods and Properties

This section lists the methods and properties, as well as returns (if applicable) for the Verity search classes.

Note. Search class objects are instantiated from both the Session and PortalRegisty objects.

SearchQuery Methods

Method	*Returns*
Execute*(Start,Size)*	SearchResult collection
Parse()	ParseResult collection

SearchQuery Properties

Property	*Returns*
HitCount	Number ^{RO}
Indexes	String
KnowledgeBase	String
Language	String
ProcessedCount	Number ^{RO}
QueryText	String
RequestedFields	String

Property	***Returns***
ScorePrecision	String
SortSpecifications	String

ParseResult Collection Methods

Method	***Returns***
First()	ParseResult object
Next()	ParseResult object or Null

ParseResult Collection Properties

Property	Returns
ErrorCount	Number RO
WarningCount	Number RO

ParseResult Properties

Property	***Returns***
Message	String RO
Severity	String RO

SearchResult Collection Methods

Method	***Returns***
First()	SearchResult
Item*(number)*	SearchResult

Method	*Returns*
Next()	SearchResult

SearchResult Collection Property

Property	*Returns*
Count	Number ^{RO}

SearchResult Properties

Property	*Returns*
Key	String ^{RO}
Score	String ^{RO}
ScoreAsNumber	Floating point number ^{RO}
SearchFields	SearchField collection ^{RO}

SearchField Collection Methods

Method	*Returns*
First()	SearchField
ItemByName*(Name)*	SearchField
Next()	SearchField

SearchField Collection Property

Property	*Returns*
Count	Number ^{RO}

SearchField Properties

Property	***Returns***
Name	String RO
Value	String RO

SearchIndexes Collection Methods

Method	***Returns***
First()	SearchIndex object
ItemByName*(Name)*	SearchIndex object
Next()	SearchIndex object

SearchIndexes Collection Property

Property	***Returns***
Count	Number RO

SearchIndex Method

Method	***Returns***
Save()	Boolean

SearchIndex Properties

Property	Returns
ExtraOptions	Any
FSOpts	Search FS Options object RO

HTTPOpts	Search HTTP Options object RO
Languages	Search Language collection RO
Location	String
Name	String RO
RecOpts	SearchRecord Options collection RO
Schedules	Search Schedule collection RO
Type	String

SearchRecord Options Properties

Property	***Returns***
Fields	SearchRecordField collection RO
Filter	String
IncrementalView	String
RecName	String RO
VeggieKey	String
ZoneOptions	String

SearchRecordField Collection Methods

Method	***Returns***
DeleteItem*(Record, Field)*	Boolean
First()	SearchRecordField object

Method	***Returns***
InsertItem*(Record, Field)*	SearchRecordField object
ItemByName*(Record, Field)*	SearchRecordField object
Next()	SearchRecordField object

SearchRecordField Collection Property

Property	***Returns***
Count	Number RO

SearchRecordField Properties

Property	***Returns***
FieldName	String RO
IsAttachment	Boolean
IsVerityField	Boolean
IsWordIndex	Boolean
RecordName	String RO

SearchLanguage Collection Methods

Method	***Returns***
DeleteItem*(Name)*	Boolean
First()	Search Language object
InsertItem*(LanguageCode, MapLanguageCode)*	Search Language object

Method	*Returns*
ItemByName*(LanguageCode)*	Search Language object
Next()	Search Language object

SearchLanguage Collection Property

Property	*Returns*
Count	Number RO

SearchLanguage Properties

Property	*Returns*
LanguageCd	String RO
MapLanguageCd	String RO

SearchSchedule Collection Methods

Method	*Returns*
DeleteItem*(Name)*	Boolean
First()	Search Schedule object
InsertItem*(RunCntrlID)*	Search Schedule object
ItemByName*(RunCntrlID)*	Search Schedule object
Next()	Search Schedule object

SearchSchedule Collection Property

Property	***Returns***
Count	Number RO

SearchSchedule Properties

Property	***Returns***
BuildType	String
RunCntrID	String RO
RunRecurrance	String
ServerName	String

SearchHTTPOptions Properties

Property	***Returns***
AllowHTTPS	Boolean
DomainLimit	String
GlobList	String
GlobListType	String
LinkDepth	Number
MIMEList	String
MIMEListType	String
ProxyHost	String
ProxyPort	Number

Property	***Returns***
StartOpts	Search Start Options collection [RO]

SearchFSOptions Properties

Property	***Returns***
GlobList	String
GlobListType	String
MIMEList	String
MIMEListType	String
StartOpts	Search Start Options collection [RO]

SearchStartOptions Collection Methods

Method	***Returns***
DeleteItem*(Value)*	Boolean
First()	Search Start Options object
InsertItem*(Value)*	Search Start Options object
ItemByName*(Value)*	Search Start Options object
Next()	Search Start Options object

SearchStartOptions Collection Property

Property	***Returns***
Count	Number [RO]

SearchStartOptions Properties

Property	***Returns***
IsDomainRestricted	Boolean
IsHostRestricted	Boolean
Value	String RO

PortalRegistry Classes Methods and Properties

This section lists the methods, and properties, as well as returns (if applicable) for the PortalRegistry classes.

PortalRegistry Methods

Method	***Returns***
BuildSearchIndex*(Language)*	Boolean
Close()	Boolean
CopyObject*(sourcePortalName,sourceRefType, sourceObjName,targetPortalName, targetPrntFldrName,copyChildren)*	Boolean
Create*(RegistryName)*	Boolean
CreateContentRefLink*(LinkName,LinkLabel, LinkParent,CRefPortalName, CRefObjectName)*	ContentReference link object
CreateRemote*(PortalName,RemoteNodeName)*	Boolean
Delete*(RegistryName)*	Boolean
DeleteHomepage()	Boolean
FindCRefByName*(Name)*	ContentReference object

Method	***Returns***
FindCRefByURL*(URL)*	ContentReference object
FindCrefForURL*(URL)*	ContentReference object
FindCRefLinkByName*(LinkName)*	ContentReference link object
FindFolderByName*(Name)*	Folder object
FindPgltByName*(PageletName)*	Pagelet object
GetAbsoluteContentURL*(NodeName,URL)*	ContentReference object
GetDefaultHPTabOID()	String
GetQualifiedURL*(ContentProvider,RelativeURL)*	URL string
GetSearchQuery()	SearchQuery object
GrantPermissionForComponent*(MenuName, ComponentName,Market,PermListName, NodeName)*	Boolean
GrantPermissionForScript*(RecordName,FieldName, EventName,FuncName, PermListName,NodeName)*	Boolean
Open*(RegistryName)*	Boolean
PermissionListDelete*(PermListName)*	Boolean
PermissionListSaveAs*(PermListSourceName, PermListTargetName)*	Boolean
RevokePermissionForComponent*(MenuName, ComponentName,Market,PermListName, NodeName)*	Boolean
RevokePermissionForScript*(RecordName,FieldName, EventName,FuncName, PermListName,NodeName)*	Boolean
Save()	Boolean

PortalRegistry Properties

Property	***Returns***
DefaultTemplate	String
Description	String
DisableFolderNavigation	Boolean
Favorites	Favorite Collection ^{RO}
FolderNavObject	String
Homepage	Homepage object ^{RO}
IsFolderNavigation	Booldan
Name	String ^{RO}
NodeTemplates	NodeTemplate Collection ^{RO}
OwnerId	String
PageletCategories	PageletCategories Collection ^{RO}
Portals	Portal collection ^{RO}
RootFolder	Folder object ^{RO}
TabDefinitions	TabDefinitions Collection ^{RO}
TemplateObject	ContentReference object ^{RO}

PortalRegistry Collection Methods

Method	***Returns***
First()	PortalRegistry object
Item*(number)*	PortalRegistry object
Next()	PortalRegistry object

PortalRegistry Property

Property	***Returns***
Count	Number ^{RO}

Node Properties

Property	***Returns***
ActiveNode	Boolean ^{RO}
AppRelease	String ^{RO}
ContentURI	String ^{RO}
DefaultPortalName	String ^{RO}
Description	String ^{RO}
Name	String ^{RO}
NodePassword	String ^{RO}
NodeType	String ^{RO}
PortalURI	String ^{RO}
ToolsRelease	String ^{RO}

Node Collection Methods

Method	***Returns***
First()	Node object
ItemByName*(NodeName)*	Node object
Next()	Node object

Node Collection Property

Property	***Returns***
Count	Number ^{RO}

Remote Node Collection Methods

Method	***Returns***
First()	Node object
ItemByName*(NodeName)*	Node object
Next()	Node object

Remote Node Collection Property

Property	***Returns***
Count	Number ^{RO}

Portal Method

Method	***Returns***
Save()	Boolean

Portal Properties

Property	Returns
HostNodeName	String
IsLocal	Boolean RO
Name	String RO

Portal Collection Methods

Method	Returns
First()	Portal object
ItemByName*(PortalName)*	Portal object
Next()	Portal object

Portal Collection Property

Property	Returns
Count	Number RO

Node Template Properties

Property	Returns
DefaultTemplate	String
Name	String
TemplateObject	Template object RO

Node Template Collection Methods

Method	***Returns***
DeleteItem*(NodeName)*	Boolean
First()	NodeTemplate object
InsertItem*(NodeName)*	NodeTemplate object
ItemByName*(NodeName)*	NodeTemplate object
Next()	NodeTemplate Object

Node Template Collection Property

Property	***Returns***
Count	Number ^{RO}

Folder Method

Method	***Returns***
Save()	Boolean

Folder Properties

Property	***Returns***
AbnContentProvider	String
AbnDataSource	String
AbnPeopleCode	String
Attributes	Attribute collection object ^{RO}
Author	String ^{RO}

Property	***Returns***
AuthorAccess	Boolean
Authorized	Boolean
CascadedPermissions	PermissionValue collection ^{RO}
CascadedRolePermissions	RolePermissionValue collection ^{RO}
ContentRefs	ContentReference Collection ^{RO}
CreationDate	String ^{RO}
DefaultChartNavigation	Boolean
Description	String
Folders	Folder collection ^{RO}
IsVisible	Boolean ^{RO}
Label	String
Name	String ^{RO}
OwnerID	String
ParentName	String ^{RO}
Path	String ^{RO}
Permissions	PermissionValue collection ^{RO}
Product	String
PublicAccess	Boolean

Property	*Returns*
RolePermissions	RolePermissionValue collection ^{RO}
SequenceNumber	Number
TreeEffectiveDate	String
TreeName	String
TreeSetId	String
TreeStructureName	String
TreeUserKeyValue	String
ValidFrom	String
ValidTo	String

Folder Collection Methods

Method	*Returns*
DeleteItem*(FolderName)*	Boolean
First()	Folder object
InsertItem*(FolderName,Label)*	Folder object
ItemByName*(FolderName)*	Folder object
Next()	Folder object

Folder Collection Property

Property	*Returns*
Count	Number ^{RO}

ContentReference Methods

Method	*Returns*
CreateLink*(Name,Label)*	ContentReference link object
Save()	Boolean

ContentReference Properties

Property	*Returns*
AbsoluteContentURL	String ^{RO}
AbsolutePortalURL	String ^{RO}
AssignedPagelets	AssignedPagelets collection ^{RO}
Attributes	Attribute collection object ^{RO}
Author	String ^{RO}
AuthorAccess	Boolean
Authorized	Boolean
CascadedPermissions	PermissionValue collection ^{RO}
CascadedRolePermissions	RolePermissionValue collection ^{RO}
ContentProvider	String
CreationDate	String ^{RO}

Property	***Returns***
Data	String
Description	String
HtmlText	String RO
IsVisible	Boolean RO
Label	String
Links	Link collection RO
Name	String RO
OwnerId	String
ParentName	String RO
Path	String RO
Permissions	PermissionValue collection RO
Product	String
PublicAccess	Boolean
RelativeURL	String RO
RolePermissions	RolePermissionValue collection RO
SequenceNumber	Number
StorageType	String
Template	String

Property	***Returns***
TemplateObject	ContentReference object RO
TemplateType	String
URL	String
URLType	String
UsageType	String
ValidFrom	String
ValidTo	String

ContentReference Collection Methods

Method	***Returns***
DeleteItem*(ContentReferenceName)*	Boolean
First()	ContentReference object
InsertItem*(ContentReferenceName, ContentReferenceLabel, Node, URL)*	ContentReference object
ItemByName*(ContentReferenceName)*	ContentReference object
Next()	ContentReference object

ContentReference Collection Property

Property	***Returns***
Count	Number RO

AttributeValue Properties

Property	***Returns***
Label	String
Name	String RO
Translatable	Boolean
Value	String

AttributeValue Collection Methods

Method	***Returns***
DeleteItem*(AttributeValueName)*	Boolean
First()	AttributeValue object
InsertItem*(AttributeValueName)*	AttributeValue object
ItemByName*(AttributeValueName)*	AttributeValue object
Next()	AttributeValue object

AttributeValue Collection Property

Property	***Returns***
Count	Number RO

PermissionValue Properties

Property	***Returns***
Cascade	Boolean
Name	String

PermissionValue Collection Methods

Method	Returns
DeleteItem*(PermissionValueName)*	Boolean
First()	PermissionValue object
InsertItem*(PermissionValueName)*	PermissionValue object
ItemByName*(PermissionValueName)*	PermissionValue object
Next()	PermissionValue object

PermissionValue Collection Property

Property	Returns
Count	Number [RO]

RolePermissionValue Collection Methods

Method	Returns
DeleteItem*(PermissionValueName)*	Boolean
First()	PermissionValue object
InsertItem*(PermissionValueName)*	PermissionValue object
ItemByName*(PermissionValueName)*	PermissionValue object
Next()	PermissionValue object

RolePermissionValue Collection Property

Property	Returns
Count	Number RO

ContentReference Link Methods

Method	Returns
Delete()	Boolean
Save()	Boolean

ContentReference Link Properties

Property	Returns
AbsoluteContentURL	String RO
AbsolutePortalURL	String RO
Attributes	Attribute Collection
Author	String RO
AuthorAccess	Boolean RO
Authorized	Boolean RO
CascadedPermissions	PermissionValue collection RO
ContentProvider	String RO
CreationDate	String RO
Data	String
Description	String
IsVisible	Boolean RO
Label	String
Name	String
OwnerId	String
ParentName	String

Property	Returns
Path	String RO
Permissions	PermissionValue Collection RO
Product	String
PublicAccess	Boolean RO
RelativeURL	String RO
SequenceNumber	String
Template	String
TemplateObject	String RO
TemplateType	String
URL	String
URLType	String RO
ValidFrom	Date
ValidTo	Date

Link Collection Methods

Method	Returns
First	Link object
Next	Link object

Link Collection Property

Property	Returns
Count	Number RO

Link Class Properties

Property	Returns
LinksObjectName	String RO

Property	Returns
LinksObjectType	String RO
LinksPortalName	String RO

TabDefinition Method

Method	Returns
Save()	Boolean

TabDefinition Properties

Property	Returns
AssignedPagelets	AssignedPagelets collection RO
AvailableCategories	AvailableCategories collection RO
AvailablePagelets	AvailablePagelets collection RO
Attributes	AttributeValue collection object RO
Author	String RO
AuthorAccess	Boolean
Authorized	Boolean
ColumnLayout	String
CreationDate	String RO
Description	String
DynamicCategories	DynamicCategories collection RO
HelpID	String

Property	***Returns***
HtmlText	String RO
IsHideActionBar	Boolean
IsLayoutLocked	Boolean
IsRenamable	Boolean
Label	String
LayoutBehavior	String
Name	String RO
OwnerId	String
Product	String
PublicAccess	Boolean
QualifiedURL	String RO
SequenceNumber	Number
StyleSheet	String
ValidFrom	String
ValidTo	String

TabDefinition Collection Methods

Method	***Returns***
DeleteItem*(TabDefinitionName)*	Boolean

Method	***Returns***
First()	TabDefinition object
InsertItem*(TabDefinitionName)*	TabDefinition object
ItemByName*(TabDefinitionName)*	TabDefinition object
Next()	TabDefinition object

TabDefinition Collection Property

Property	***Returns***
Count	Number ^{RO}

AssignedPagelet Properties

Property	***Returns***
Column	Number
LayoutBehavior	String
PageletName	String ^{RO}
Row	Number

AssignedPagelet Collection Methods

Method	***Returns***
DeleteItem*(PageletName)*	Boolean
First()	Pagelet object
InsertItem*(PageletName,Column,Row,LayoutBehavior)*	Pagelet object

Method	Returns
ItemByName*(PageletName)*	Pagelet object
Next()	Pagelet object

AssignedPagelet Collection Property

Property	Returns
Count	Number RO

AvailableCategory Properties

Property	Returns
AvailablePagelets	AvailablePagelets Collection RO
CategoryName	String RO

AvailableCategory Collection Methods

Method	Returns
First()	AvailableCategory object
ItemByName*(Name)*	AvailableCategory object
Next()	AvailableCategory object

AvailableCategory Collection Property

Property	Returns
Count	Number RO

AvailablePagelet Properties

Property	Returns
CategoryLabel	String RO
CategoryName	String RO
Column	Number RO
LayoutBehavior	String RO
PageletLabel	String RO
PageletName	String RO
Row	Number RO

AvailablePagelet Collection Methods

Method	Returns
First()	AvailablePagelet object
ItemByName*(Name)*	AvailablePagelet object
Next()	AvailablePagelet object

AvailablePagelet Collection Property

Property	Returns
Count	Number RO

DynamicCategory Collection Methods

Method	Returns
DeleteItem*(Name)*	Boolean

Method	*Returns*
First()	String
InsertItem*(Name)*	String
ItemByName*(Name)*	String
Next()	String

DynamicCategory Collection Property

Property	*Returns*
Count	Number [RO]

PageletCategory Method

Method	*Returns*
Save()	Boolean

PageletCategory Properties

Property	*Returns*
Attributes	AttributeValues collection object [RO]
Author	String [RO]
AuthorAccess	Boolean
Authorized	Boolean
CascadedPermissions	PermissionList object [RO]
CreationDate	String [RO]

Property	*Returns*
Description	String
Label	String
Name	String RO
OwnerId	String
Pagelets	Pagelets Collection RO
Permissions	PermissionList Collection RO
Product	String
PublicAccess	Boolean
SequenceNumber	Number

PageletCategory Collection Methods

Method	*Returns*
DeleteItem*(Name)*	Boolean
First()	String
InsertItem*(Name,Label)*	String
ItemByName*(Name)*	String
Next()	String

PageletCategory Collection Property

Property	*Returns*
Count	Number [RO]

Pagelet Method

Method	*Returns*
Save()	Boolean

Pagelet Properties

Property	*Returns*
Attributes	Attribute collection object [RO]
Author	String [RO]
AuthorAccess	Boolean
Authorized	Boolean
CascadedPermissions	PermissionList Collection [RO]
ContentProvider	Node object [RO]
CreationDate	String [RO]
DefaultColumn	Number
Description	String
HelpID	String
IsHideMinimize	Boolean
Label	String

Property	***Returns***
Name	String ^{RO}
OwnerId	String
ParentName	String ^{RO}
Permissions	PermissionList Collection ^{RO}
Product	String
PublicAccess	Boolean
QualifiedURL	String ^{RO}
SequenceNumber	Number
URL	String
URLType	String

Pagelet Collection Methods

Method	***Returns***
DeleteItem*(Name)*	Boolean
First()	Pagelet object
InsertItem*(Name,Label,NodeName,URL)*	Pagelet object
ItemByName*(Name)*	Pagelet object
Next()	Pagelet object

Pagelet Collection Property

Property	Returns
Count	Number RO

UserHomepage Method

Method	Returns
Save()	

UserHomepage Properties

Property	Returns
Greeting	String
UserId	String RO
UserTab	UserTab CollectionRO

UserTab Properties

Property	Returns
ColumnLayout	Number
Label	String
QualifiedURL	String RO
SelectedPagelets	SelectedPagelets collectionRO
SequenceNumber	Number
TabName	String RO

UserTab Collection Methods

Method	***Returns***
DeleteItem*(Name)*	Boolean
First()	UserTab object
InsertItem*(Name)*	UserTab object
ItemByName*(Name)*	UserTab object
Next()	UserTab object

UserTab Collection Property

Property	***Returns***
Count	Number ^{RO}

SelectedPagelet Properties

Property	***Returns***
CategoryName	String ^{RO}
Column	Number
IsMinimized	Boolean
PageletName	String ^{RO}
Row	Number

SelectedPagelet Collection Methods

Method	***Returns***
DeleteItem*(Name)*	Boolean

Method	***Returns***
First()	SelectedPagelet object
InsertItem*(Name)*	SelectedPagelet object
ItemByName*(Name)*	SelectedPagelet object
Next()	SelectedPagelet object

SelectedPagelet Collection Property

Property	***Returns***
Count	Number RO

Favorites Properties

Property	***Returns***
CRefName	String RO
Label	String
QualifiedURL	String RO
SequenceNumber	Number
URL	String

Favorites Collection Methods

Method	***Returns***
DeleteItem*(FavoriteLabel)*	Boolean
First()	Favorite object

Method	***Returns***
InsertItem*(FavoriteLabel,FavoriteName)*	Favorite object
ItemByLabel*(FavoriteLabel)*	Favorite object
Next()	Favorite object

Favorites Collection Property

Property	***Returns***
Count	Number RO

Query Classes Methods and Properties

This section lists the functions, methods, and properties, as well as returns (if applicable) for the query classes.

Query Collection Methods

Method	***Returns***
First()	Query object
Item*(Number)*	Query object
ItemByName*(Name)*	Query object
Next()	Query object

Query Collection Property

Property	***Returns***
Count	Number RO

Query Methods

Method	***Returns***
AddPrompt*(PromptName)*	Prompt object
AddQuerySelect()	QuerySelect object
AddTrackingURL*(URLString)*	None
Close()	None
CopyPrivateQuery*(QueryName,QryType,TargetUserId)*	Number
Create*(QueryName,Pubic,Type,Description, LongDescription*)	Query object
Delete()	Number
DeletePrompt*(PromptName)*	Number
FindExpression*(Number)*	Expression object
FormatBinaryResultString*(&Rowset,Output_Format, StartRow,EndRow*)	Binary object
FormatResultString*(&Rowset,Output_Format,StartRow, EndRow*)	String
GetTreePromptCount()	Number
Open(*QueryName,Public,Update)*	Number
Rename*(NewQueryName)*	Number
RunToFile*(&PromptRecord,Destination,OutputFormat, MaxRows*)	Number
RunToRowset*(&PromptRecord,MaxRows)*	Rowset object
RunToString *(&PromptRecord,ChunkSize, OutputFormat,MaxRows*)	String

Method	***Returns***
Save()	Number
SetTrackingURL(*ExpressionText*,*ExpressionNumber)*	None

Query Properties

Property	***Returns***
Approved	String
ApprovedDtTm	String RO
ApprovedUserId	String
CreateDtTm	String RO
CreateUserId	String RO
Description	String
ExecAppName	String
ExecLogging	Boolean
LastSQLErrorCode	Number RO
LastUpdDttm	String RO
LastUpdOprId	String RO
LongDescription	String
Metadata	Metadata collection RO
MoreRowsAvailable	Boolean RO

Property	***Returns***
Name	String RO
OutputUnicode	Boolean
PDFFont	String
Prompts	QueryPrompt collection RO
PromptRecord	Record object RO
PublicPrivate	String
QuerySelect	QuerySelect object RO
QueryStatistics	QueryStatistics object RO
RunTimePrompts	QueryPrompt Collection RO
SQL	String RO
Type	Number

QuerySelect Methods

Method	***Returns***
AddAllFields*(QueryRecord)*	Number
AddCriteria*(Name)*	QueryCriteria object
AddExpression*(Name)*	QueryExpression object
AddHavingCriteria*(Name)*	QueryCriteria object
AddQueryOutputField*(QueryRecord,index)*	QueryOutputField object

Method	***Returns***
AddQueryRecord*(QueryRecordName)*	QueryRecord object
AddQuerySelectedField*(QueryRecord,index)*	QuerySelectedField object
DeleteCriteria*(index)*	Number
DeleteExpression*(Index)*	Number
DeleteField*(Index)*	Number
DeleteHavingCriteria*(Index)*	Number
DeleteRecord*(Index)*	Number

QuerySelect Properties

Property	***Returns***
Criteria	QueryCriteria collection ^{RO}
Distinct	Boolean
Expressions	QueryExpression collection ^{RO}
HavingCriteria	QueryCriteria collection ^{RO}
ParentSelectNum	Number ^{RO}
QueryOutputFields	QueryField collection ^{RO}
QueryRecords	QueryRecord collection ^{RO}
QuerySelectedFields	QueryField collection ^{RO}
QuerySelects	QuerySelect object ^{RO}

Property	*Returns*
SelectNum	Number RO
SelectType	Number RO

QuerySelect Collection Methods

Method	*Returns*
AddUnion()	QuerySelect object
DeleteQuerySelect*(SelNumber)*	Number
First()	QuerySelect object
Item*(Number)*	QuerySelect object
ItemBySelNum*(SelNumber)*	QuerySelect object
Next()	QuerySelect object

QuerySelect Collection Properties

Property	*Returns*
Count	Number RO

QueryRecord Collection Methods

Method	*Returns*
First()	QueryRecord object
Item*(Number)*	QueryRecord object
ItemByAlias*(Alias)*	QueryRecord object

Method	***Returns***
Next()	QueryRecord object

QueryRecord Collection Property

Property	***Returns***
Count	Number ^{RO}

QueryRecord Method

Method	***Returns***
GetField*(Index)*	QueryField object

QueryRecord Properties

Property	***Returns***
Description	String ^{RO}
JoinAlias	String
JoinFieldName	String
JoinType	Number
Name	String ^{RO}
QueryFields	QueryFields collection ^{RO}
RecordAlias	String

QueryField Collection Methods

Method	***Returns***
First()	QueryField object
Item*(Number)*	QueryField object
ItemByNameAndAlias*(Name,RecordAlias)*	QueryField object
ItemByExpNum*(ExpNum)*	QueryField object
Next()	QueryField object

QueryField Collection Property

Property	***Returns***
Count	Number RO

QueryField Methods

Method	***Returns***
AddTranslateExpression*(ExpressionName)*	QueryExpression object
AddTranslateField*(FieldName)*	QueryField object
GetImageFormat()	Number

QueryField Properties

Property	***Returns***
Aggregate	Number
ColumnNumber	Number
Description	String RO

Property	***Returns***
Decimal	Number [RO]
ExpNum	Number
Flag	Number [RO]
Format	Number
HeadingText	String
HeadingType	String
HeadingUniqueFieldName	String
Length	Number [RO]
LongName	String [RO]
Name	String [RO]
OrderByDirection	Number
OrderByNumber	Number
QueryRecord	QueryRecord object [RO]
RecordAlias	String [RO]
ShortName	String [RO]
TranslateEffDtLogic	String
TranslateExpression	Expression object [RO]
TranslateField	QueryField [RO]

Property	*Returns*
TranslateOption	String
Type	String

QueryCriteria Collection Methods

Method	*Returns*
First()	QueryCriteria object
Item*(Number)*	QueryCriteria object
ItemByName*(CriteriaName)*	QueryCriteria object
Next()	QueryCriteria object

QueryCriteria Collection Property

Property	*Returns*
Count	Number ^{RO}

QueryCriteria Methods

Method	*Returns*
AddExpr1Expression()	QueryExpression object
AddExpr1Field*(QueryRecordAlias,FieldName)*	QueryField object
AddExpr2Expression()	QueryExpression object
AddExpr2Field1*(QueryRecordAlias,FieldName)*	QueryField object
AddExpr2Field2*(QueryRecordAlias,FieldName)*	QueryField object

Method	***Returns***
AddExpr2List()	QueryList object
AddExpr2Subquery()	QuerySelect object

QueryCriteria Properties

Property	***Returns***
Expr1Expression	QueryExpression object
Expr1Field	QueryField object ^{RO}
Expr1Type	Number
Expr2Constant1	String
Expr2Constant2	String
Expr2Expression1	QueryExpression object
Expr2Expression2	QueryExpression object
Expr2Field1	QueryField object ^{RO}
Expr2Field2	QueryField object ^{RO}
Expr2List	QueryList object ^{RO}
Expr2Subquery	QuerySelect object ^{RO}
Expr2Type	Number
Logical	Number
LParenLvl	Number

Property	*Returns*
Name	String RO
Negation	Boolean
Operator	Number
R1ExprNum	Number
R2ExprNum	Number
R1ExprType	Number
R2ExprType	Number
RParenLvl	Number

QueryExpression Collection Methods

Method	*Returns*
First()	QueryExpression object
Item*(Number)*	QueryExpression object
ItemByName*(ExpressionName)*	QueryExpression object
Next()	QueryExpression object

QueryExpression Collection Property

Property	*Returns*
Count	Number RO

QueryExpression Properties

Property	***Returns***
Aggregate	Number
BindFlag	Number
Decimal	Number
ExpNum	Number
Length	Number
Name	String ^{RO}
OutputField	QueryField object
RightExprFlag	Number
SelectedField	QueryField object
Text	String
Type	Number

QueryList Methods

Method	***Returns***
AddListValue*(Value,IsPrompt)*	QueryListValue object
First()	QueryListValue object
Item*(Number)*	QueryListValue object
Next()	QueryListValue object

QueryList Property

Property	*Returns*
Count	Number RO

QueryListValue Properties

Property	*Returns*
IsPrompt	Boolean RO
Value	String RO

QueryRecordHierarchy Collection Methods

Method	*Returns*
First()	QueryRecordHierarchy object
Item*(Number)*	QueryRecordHierarchy object
ItemByName*(Name)*	QueryRecordHierarchy object
Next()	QueryRecordHierarchy object

QueryRecordHierarchy Collection Property

Property	*Returns*
Count	Number RO

QueryRecordHierarchy Properties

Property	*Returns*
Description	String RO
Level	Number RO

Property	*Returns*
Name	String ^{RO}
ParentFlag	Number ^{RO}

Query Metadata Collection Methods

Method	*Returns*
First()	Query Metadata object
Item*(Number)*	Query Metadata object
ItemByName*(Name)*	Query Metadata object
Next()	Query Metadata object

Query Metadata Collection Property

Property	*Returns*
Count	Number ^{RO}

Query Metadata Properties

Property	*Returns*
Name	String ^{RO}
Value	String ^{RO}

QueryStatistics Properties

Property	*Returns*
AvgExecTime	Number ^{RO}

Property	***Returns***
AvgFetchTime	Number RO
AvgNumRows	Number RO
ExecCount	Number RO
LastExecDtTm	String RO

QuerySecurityProfile Properties

Property	***Returns***
AllowAnyJoin	Boolean RO
AllowDistinct	Boolean RO
AllowExpressions	Boolean RO
AllowSubqueries	Boolean RO
AllowUnions	Boolean RO
ApprovePrivateQuery	Boolean RO
ApprovePublicQuery	Boolean RO
CanCreatePublic	Boolean RO
CanCreateWorkFlow	Boolean RO
CanModifyQuery	Boolean RO
CanRunQuery	Boolean RO
CanRunToCrystal	Boolean RO

Property	*Returns*
CanRunToExcel	Boolean ^{RO}
LimitUnapproved	Boolean ^{RO}
MaxInTreeCriteria	Boolean ^{RO}
MaxJoins	Boolean ^{RO}
MaxRowsToFetch	Boolean ^{RO}
MaxUnapprovedRows	Boolean ^{RO}

QueryDBRecord Collection Methods

Method	*Returns*
First()	QueryDBRecord object
Item*(Number)*	QueryDBRecord object
ItemByName*(Name)*	QueryDBRecord object
Next()	QueryDBRecord object

QueryDBRecord Collection Property

Property	*Returns*
Count	Number ^{RO}

QueryDBRecord Methods

Method	*Returns*
QueryDBRecordFieldByIndex*(Index)*	QueryDBRecordField object

Method	*Returns*
QueryDBRecordFieldByName*(Name)*	QueryDBRecordField object

QueryDBRecord Properties

Property	*Returns*
Description	String RO
Name	String RO
QueryDBRecordFields	QueryDBRecordFields collection
RecordHierarchy	QueryRecordHierarchy Collection RO

QueryDBRecordField Collection Methods

Method	*Returns*
First()	QueryDBRecordField object
Item*(Number)*	QueryDBRecordField object
ItemByName*(Name)*	QueryDBRecordField object
Next()	QueryDBRecordField object
Sort*(SortCriteria)*	0

QueryDBRecordField Collection Property

Property	*Returns*
Count	Number RO

QueryDBRecordField Method

Method	***Returns***
GetImageFormat()	Number

QueryDBRecordField Properties

Property	***Returns***
Decimal	Number RO
Description	String RO
Flag	Number RO
Format	Number RO
Length	Number RO
LongName	String RO
LookupTableName	String RO
LookupTableRecord	QueryDBRecord object RO
Name	String RO
ShortName	String RO
Type	Number RO

QueryPrompt Collection Methods

Method	***Returns***
First()	QueryPrompt object
Item*(Number)*	QueryPrompt object

Method	***Returns***
ItemByName*(Name)*	QueryPrompt object
Next()	QueryPrompt object

QueryPrompt Collection Property

Property	***Returns***
Count	Number RO

QueryPrompt Properties

Property	***Returns***
EditType	Number
FieldDecimal	Number
FieldFormat	Number
FieldLength	Number
FieldName	String
FieldType	Number
HeadingText	String
HeadingType	Number
LangCount	Number RO
Name	String RO
PromptRecordFieldName	String RO

Property	*Returns*
PromptTable	String
UniquePromptName	String
UseCount	Number RO

Tree Classes Methods and Properties

This section lists the methods and properties, as well as returns (if applicable) for the tree classes.

Branch Collection Properties

Property	*Returns*
Count	Number RO

Leaf Methods

Method	*Returns*
Cut()	Number
Delete()	Number
DeleteByRange*(RangeFrom,RangeTo)*	Number
GenABNMenuElement()	String
InsertDynSib()	Number
InsertSib*(RangeFrom,RangeTo)*	Leaf object
LoadABNChart*(&chart_rowset,&relactions_rowset*	None
LoadABNChartOrdered*(&chart_rowset, &relactions_rowset,order*	None

Method	***Returns***
MoveAsChild(*Node)*	Number
MoveAsChildByName*(NodeName)*	Number
MoveAsSib*(Leaf)*	Number
MoveAsSibByRange*(RangeFrom,RangeTo)*	Number
PasteSib()	Number
RefreshDescription*(expand_range)*	Number
UpdateRanges*(RangeFrom,RangeTo)*	Number

Leaf Properties

Property	***Returns***
Description	String RO
DisplayLevelNumber	Number RO
Dynamic	Boolean
HasNextSib	Boolean RO
HasPrevSib	Boolean RO
ImageName	String
IsChanged	Boolean RO
IsCut	Boolean RO
IsDeleted	Boolean RO
IsInserted	Boolean RO
NextSib	Leaf object RO
Parent	Node object RO
PrevSib	Leaf object RO

Property	Returns
RangeFrom	String
RangeTo	String
TreeBranchName	String RO
TreeEffDt	Date
TreeName	String RO
TreeSetID	String RO
TreeUserKeyValue	String RO

Level Collection Methods

Method	Returns
Add(*LevelName)*	Level object
Item*(LevelName,LevelNumber)*	Level object
Remove()	Number

Level Collection Properties

Property	Returns
Count	Number RO
First	Level object RO
Last	Level object RO
Next	Level object RO

Level Properties

Property	***Returns***
AllValuesAudit	Boolean
Description	String
Name	String
Number	Number
TreeBranchName	String RO
TreeEffDt	Date
TreeName	String RO
TreeSetID	String RO
TreeUserKeyValue	String RO

Node Methods

Method	***Returns***
Branch()	Number
Cut()	Number
Delete()	Number
DeleteByName*(NodeName)*	Number
Expand*(ExpandType)*	Number
GenABNMenuElement*(initial_node)*	String
GenBreadCrumbs*(list)*	String
GenRelatedActions()	String
InsertChildLeaf*(RangeFrom,RangeTo)*	Leaf object
InsertChildNode*(NodeName)*	Node object

Method	***Returns***
InsertChildRecord*(NodeName)*	Node object
InsertDynChildLeaf()	Leaf object
InsertSib*(NodeName)*	Node object
InsertSibRecord*(NodeName)*	Node object
LoadABNChart*(&chart_rowset,&relactions_rowset, requested_node,initial_node)*	None
LoadABNChartOrdered*(&chart_rowset, &relactions_rowset,requested_node, initial_node,order)*	None
MoveAsChild*(Node)*	Number
MoveAsChildByName*(NodeName)*	Number
MoveAsSib*(Node)*	Number
MoveAsSibByName*(NodeName)*	Number
PasteChild()	Number
PasteSib()	Number
RefreshDescription()	Number
Rename*(NodeName)*	Number
SwitchLevel*(NewLevelNumber)*	Number
Unbranch()	Number

Node Properties

Property	*Returns*
AllChildCount	Number [RO]
AllChildNodeCount	Number [RO]
ChildLeafCount	Number [RO]
ChildNodeCount	Number [RO]
ColImageName	String
Description	String [RO]
DisplayLevelNumber	Number [RO]
ExpImageName	String
FirstChildLeaf	Leaf object [RO]
FirstChildNode	Node object [RO]
HasChildLeaves	Boolean [RO]
HasChildNodes	Boolean [RO]
HasChildren	Boolean [RO]
HasNextSib	Boolean [RO]
HasPrevSib	Boolean [RO]
IsBranched	Boolean [RO]
IsChanged	Boolean [RO]
IsCut	Boolean [RO]
IsDeleted	Boolean [RO]
IsInserted	Boolean [RO]
IsRoot	Boolean [RO]
LastChildLeaf	Leaf object [RO]
LastChildNode	Node object [RO]
LevelNumber	Number
Name	String

Property	***Returns***
NextSib	Leaf object RO
Parent	Node object RO
PrevSib	Leaf object RO
State	Number RO
TreeBranchName	String RO
TreeEffDt	Date
TreeName	String RO
TreeSetID	String RO
TreeUserKeyValue	String RO
Type	String RO

Tree Methods

Method	***Returns***
Audit()	Number
AuditByName*(SetID, UserKeyValue, TreeName, EffDt, BranchName)*	Number
Close()	Number
Copy*(FromSetId, FromUserKeyValue, FromTreeName, FromEffDt, FromBranchName, ToSetId, ToUserKeyValue, ToTreeName, ToEffDt, ToBranchName)*	Number
Create*(SetID, UserKeyValue, TreeName, EffDt, StructureName)*	Number
Delete*(SetID, UserKeyValue, TreeName, EffDt, BranchName)*	Number
Exists*(SetID, UserKeyValue, TreeName, EffDt, BranchName)*	Number

Method	***Returns***
FindLeaf*(RangeFrom,RangeTo)*	Leaf object
FindNode*(NodeName, Description)*	Node object
FindRoot()	Node object
InsertRoot*(NodeName)*	Node object
LeafExists*(RangeFrom, RangeTo)*	Number
LockTree()	Number
Open*(SetID, UserKeyValue, TreeName, EffDt, BranchName, Update)*	Number
OpenAsOfDate*(SetID, UserKeyValue, TreeName, AsOfDate, BranchName, Update)*	Number
OpenForExport*(SetID,UserKeyValue,TreeNameEffDt)*	Number.
OpenWholeTree*(SetID, UserKeyValue, TreeName, EffDt)*	Number
NodeExists*(NodeName)*	Number
Rename*(FromSetId, FromUserKeyValue, FromTreeName, FromEffDt, FromBranchName, ToTreeName)*	Number
Save()	Number
SaveAs*(SetID, UserKeyValue, TreeName, EffDt, BranchName)*	Number
SaveAsDraft*(SetID, UserKeyValue, TreeName, EffDt, BranchName)*	Number
SaveDraft()	Number

Method	***Returns***
SetImportMode*(Nodes_Number,Leaves_Number)*	Number
TreeLocksNumber*(setID,UserKeyValue,TreeName, EffDt)*	Number
UnlockTree()	Number
UpdateLock()	Number

Tree Properties

Property	***Returns***
AllValues	Boolean
AuditDetails	Boolean
Branches	Branch collection object [RO]
BranchImageName	String
BranchLevel	Number [RO]
BranchName	String [RO]
Category	String
Description	String
DuplicateLeaves	Boolean
EffDt	Date
HasDetailRanges	Boolean [RO]
HasLockedBranches	Boolean [RO]
IsBranched	Boolean [RO]
IsChanged	Boolean [RO]
IsOpen	Boolean [RO]
IsQueryTree	Boolean [RO]

Property	***Returns***
IsValid	Boolean RO
IsVersionChanged	Boolean RO
IsWholeTree	Boolean RO
KeyBranchName	String RO
KeyEffDt	String RO
KeyName	String RO
KeySetID	String RO
KeyUserKeyValue	String RO
LeafCount	Number RO
LeafImageName	String
LeafOnClipboard	Boolean RO
LevelCount	Number RO
Levels	Level collection object RO
LevelUse	String
LockOwner	String, RO
Name	String
NodeColImage	String
NodeCount	Number RO
NodeExpImage	String
NodeOnClipboard	Boolean RO
ParentLevel	Number RO
ParentName	String RO
PerformanceMethod	String
PerformanceSelector	String
PerformanceSelectorOption	String

Property	Returns
SetId	String
Status	String
Structure	Tree structure object RO
StructureName	String RO
TreeImageName	String
UserKeyValue	String
UseUpdateReservation	Boolean RO

Tree Structure Methods

Method	Returns
Close()	Number
Copy*(FromStructId, ToStructId)*	Number
Create*(StructId, Type)*	Number
Delete*(StructId)*	Number
Open*(StructId, Update)*	Number
Rename*(FromStructId, ToStructId)*	Number
Save()	Number

Tree Structure Properties

Property	Returns
Description	String
DetailComponent	String
DetailField	String

Property	***Returns***
DetailMenu	String
DetailMenuBar	String
DetailMenuItem	String
DetailMultiNavigate	Boolean
DetailPage	String
DetailRecord	String
IndirectionMethod	String
KeyName	String [RO]
LevelComponent	String
LevellMenu	String
LevelMenuBar	String
LevelMenuItem	String
LevelPage	String
LevelRecord	String
Name	String
NodeComponent	String
NodeField	String
NodelMenu	String
NodeMenuBar	String
NodeMenuItem	String
NodeMultiNavigate	Boolean
NodePage	String
NodeRecord	String
NodeUserKeyField	String
SummarySetId	String

Property	Returns
SummaryLevelNumber	String
SummaryTreeName	String
SummaryUserKeyValue	String
Type	String

Deprecated Items and PeopleCode No Longer Supported

This section discusses the following:

- Mapping of old objects to new objects.
- Deprecated products and classes.
- Deprecated functions.
- Deprecated methods and properties.
- Deprecated XML Publisher items.
- XML Publisher items no longer supported.
- Deprecated messaging PeopleCode functions, methods and properties.
- Functions no longer supported.

Mapping of Old Objects to New Objects

In PeopleTools 8.1, the names of some of the object definitions in Application Designer changed. The names of related built-in functions have changed accordingly.

The first table lists the old terms and new terms.

In the second table, the left column lists the old names of the functions, system variables, or reserved words. The right column lists the new names. The old functions, system variables, and reserved words in this table still work in PeopleTools; however, they have been deprecated and are being retained for backward compatibility only. New applications should be created using the new functions, system variables, and reserved words.

Old Term	New Term
Operator	User
Panel	Page

Old Term	***New Term***
Panel Group	Component
Business Component	Component Interface

Deprecated Function, System Variable or Reserved Word	***New Function, System Variable or Reserved Word***
DoModalPanelGroup built-in function	DoModalComponent built-in function
GetNextNumberWithGaps	GetNextNumberWithGapsCommit built-in function
IsModalPanelGroup built-in function	IsModalComponent built-in function
IsOperatorInClass built-in function	IsUserInPermissionList built-in function
PanelGroupChanged built-in function	ComponentChanged built-in function
ScheduleProcess built-in function	CreateProcessRequest built-in function
SetNextPanel built-in function	SetNextPage built-in function
TransferPanel built-in function	TransferPage built-in function
%OperatorClass System Variable	%PrimaryPermissionList System Variable
%OperatorID System Variable	%UserId System Variable
%OperatorRowLevelSecurityClass System Variable	%RowSecurityPermissionList System Variable
%Panel System Variable	%Page System Variable
%PanelGroup System Variable	%Component System Variable
PANEL reserved word	PAGE reserved word
PANELGROUP reserved word	COMPONENT reserved word

Deprecated Function, System Variable or Reserved Word	***New Function, System Variable or Reserved Word***
PanelGroup variable declaration	Component variable declaration

Business Components are now named Component Interfaces. Therefore, for Component Interfaces, the old reserved word, methods, and system variables *are no longer valid.* You *must* use the new reserved word, methods, or system variables.

No Longer Valid	***Use Instead***
COMPONENT reserved word	COMPINTFC reserved word
GetComponent method	GetCompIntfc method
FindComponent method	FindCompIntfcs method
%Component system variable	%CompIntfc system variable

Deprecated Products and Classes

The following products and their supporting classes have been deprecated:

- PeopleSoft Business Interlinks
    - Business Interlinks class
    - BIDocs class
- PeopleSoft Mobile Agent
    - Mobile class
    - PropertyInfo collection
    - CI collection
    - Data collection
    - ListViewAttrs class
    - PropertyAttrs class
    - PeerDefaultAttributes class
    - SyncServer class

Deprecated Business Interlinks Class

PeopleSoft Business Interlinks is a deprecated product. The Business Interlinks class currently exists for backward compatibility only. For new integrations, use Integration Broker instead.

This section lists the deprecated functions, methods, and properties, as well as returns (if applicable) for the Business Interlinks and BIDocs classes.

Deprecated Business Interlink Functions

Deprecated Function	***Returns***
GetBIDoc([*XMLString*])	BIDoc object
GetInterlink**(INTERLINK.***name*)	Business Interlink object

Deprecated Business Interlink Methods

Deprecated Method	***Returns***
AddInputRow*(inputname,value)*	Boolean
BulkExecute(**RECORD.***inputrec* [, RECORD.*outputrec*] [, *user_process_inst* | *user_operid*])	Number
Clear()	None
Execute()	Number
FetchIntoRecord**(RECORD.***recname*, [, *user_process_inst* | *user_operid*])	Number
FetchIntoRowset*(&Rowset)*	Number
FetchNextRow*(outputname, value)*	Output row object
GetFieldCount()	Number
GetFieldType*(index)*	Number
GetFieldValue*(index)*	String

Deprecated Method	*Returns*
InputRowset(&*Rowset)*	Number
MoveFirst()	Boolean
MoveNext()	Boolean

Deprecated Business Interlink Property

Deprecated Property	*Returns*
StopAtError	Boolean

Deprecated BIDoc Configuration Parameters

Deprecated Configuration Parameters	*Returns*
URL	String
All other configuration parameters can be accessed like read-write properties.	Depends on type of Interlink plug-in.

Deprecated BIDoc Methods

Deprecated Method	*Returns*
AddAttribute*(attributename,attributevalue)*	Number
AddComment*(comment)*	Number
AddProcessingInstruction*(instruction)*	Number
AddText*(text)*	Number
CreateElement*(elementname)*	BIDoc object
GenXMLString()	String

Deprecated Method	***Returns***
GetAttributeName*(attributenumber)*	String
GetAttributeValue({*attributenumber* | *attributename*})	String
GetNode({*nodenumber* | *nodename*})	BIDoc object

Other Business Interlinks Features That Have Been Deprecated

Other classes include functionality specifically added for the PeopleSoft Business Interlinks product. With the deprecation of PeopleSoft Business Interlinks, these features are deprecated as well.

Deprecated Item	***Returns***
Request class, GetContentBody() method	String

Deprecated Mobile Classes

PeopleSoft Mobile Agent is a deprecated product. These mobile classes currently exist for backward compatibility only.

This section lists the deprecated system variables, functions, methods, and properties, as well as returns (if applicable) for the mobile classes.

Deprecated Mobile Class System Variables

Deprecated System Variable	***Returns***
%DeviceType	The type of the mobile device
%MobilePage	The name of the current mobile page
%ThisMobileObject	Mobile object

Deprecated Mobile Class Functions

Deprecated Function	***Returns***
IsDisconnectedClient()	Boolean

Deprecated Function	***Returns***
GenerateMobileTree*(CIObject* [,*CIObject_property*])	None
MSFGetNextNumber*(Key1,Key2,Key3)*	Number
TransferMobilePage(**MOBILEPAGE.***PageName,Tab, CIObject)*	None

Deprecated Mobile Class Methods

Deprecated Method	***Returns***
Create()	Boolean
Find()	Collection of empty mobile objects
Get()	Boolean
GetName()	String
GetParent()	Mobile object
GetPropertyAttrsByName*(propertyname)*	PropertyAttrs object
GetPropertyByName*(propertyname)*	Depends on property. Will return a collection if *propertyname* is a collection.
GetPropertyInfoByName*(propertyname)*	PropertyInfo object
GetTopParent()	Mobile object
IsNew()	Boolean
IsNewAndNeverSaved()	Boolean
IsSameAs*(&MobileObject)*	Boolean
ReadBlobFromFile*(blobpropertyname,filename*	Boolean

Deprecated Method	*Returns*
Save()	Boolean
SetModified*(IsModified*	Boolean
SetPropertyByName(*propertyname)*	None
ValidateEnum*(propertyname)*	Boolean
ValuesDifferFromLastSync()	Boolean
WasSaved()	Boolean
WriteBlobToFile*(blobpropertyname,filename)*	Boolean

Deprecated Mobile Property

Deprecated Property	*Returns*
PropertyInfoCollection	PropertyInfoCollection ^{RO}

Deprecated PropertyInfo Collection Method

Deprecated Method	*Returns*
Item*(number)*	PropertyInfo object

Deprecated PropertyInfo Collection Property

Deprecated Property	*Returns*
Count	Number

Deprecated PropertyInfo Properties

Deprecated Property	*Returns*
DefaultValue	Depends on the value RO
HasDefaultValue	Boolean RO
IsCollection	Boolean RO
IsRefToParent	Boolean RO
IsPeerReference	Boolean RO
IsSimpleApplicationProperty	Boolean RO
IsSystemProperty	Boolean RO
Name	String RO
PropertyInfoCollection	PropertyInfo Collection RO

Deprecated CI Collection Method

Deprecated Method	*Returns*
Item*(index)*	Mobile object

Deprecated CI Collection Property

Deprecated Property	*Returns*
Count	Number RO

Deprecated Data Collection Methods

Deprecated Method	*Returns*
DeleteItem*(location)*	Boolean

Deprecated Method	***Returns***
GetListAttrs*(propertyname)*	ListViewAttrs object
InsertItem*(location)*	Boolean
Item*(Index)*	Mobile object

Deprecated Data Collection Property

Deprecated Property	***Returns***
Count	Number ^{RO}

Deprecated ListViewAttrs Method

Deprecated Method	***Returns***
IsSet*(attribute)*	Boolean

Deprecated ListViewAttrs Properties

Deprecated Property	***Returns***
DetailViewLabel	String
Label	String
ListViewLabel	String
Visible	Boolean

Deprecated PropertyAttrs Method

Method	***Returns***
IsSet*(attribute)*	Boolean

Deprecated PropertyAttrs Properties

Deprecated Property	***Returns***
DisplayOnly	Boolean
InError	Boolean
Label	String
ShowRequiredCue	Boolean
Visible	Boolean

Deprecated PeerDefaultAttributes Method

Deprecated Method	***Returns***
IsSet*(attribute)*	Boolean

Deprecated PeerDefaultAttributes Properties

Deprecated Property	***Returns***
DisplayOnly	Boolean
Label	String
Visible	Boolean

Other Mobile Features That Have Been Deprecated

Other classes include functionality specifically added for the PeopleSoft Mobile Agent. With the deprecation of the PeopleSoft Mobile Agent, these features are deprecated as well.

Deprecated Item	***Returns***
Record class, SystemIDFieldName property	String ^{RO}
Record class, TimeStampFieldName property	String ^{RO}

Deprecated Item	*Returns*
Folder class, IsMobile property	Boolean
ContentReference class, IsMobile property	Boolean
ContentReference link, IsMobile property	Boolean

Deprecated SyncServer Class

PeopleSoft Mobile Agent is a deprecated product. This mobile class currently exists for backward compatibility only.

This section lists the deprecated functions, methods, and properties, as well as returns (if applicable) for the SyncServer class.

Deprecated System Variable

Deprecated System Variable	*Returns*
%SyncServer	SyncServer object

Deprecated Methods

Deprecated Method	*Returns*
AddReference*(SyncID*[, *CIDefnName*])	None
AddReferenceType*(CIDefnName)*	None
CheckForChanges(*SyncDateTime*,*TotalRowCount*[, *HasRefList*])	Boolean
SelectAll()	None

Deprecated Properties

Deprecated Property	*Returns*
ClientPlatform	String
ConflictAlgorithm	String

Deprecated Property	***Returns***
ConflictStatus	String
ExcludeProperty	String
ExportLocation	String
IsReferenceUnresolved	Boolean, RO
LasySyncDateTime	DateTime, RO
LastSyncRowCount	Number, RO
PropertyValue	String
SessionID	String RO
Synchronizing	String
SyncIDs	Array of number
TypeID	String
validateID	String
validateRowCount	Number RO
validateVersion	String

Deprecated Functions

The first column in the following table lists built-in PeopleCode functions that existed prior to PeopleTools 8.0. The second column lists the new method or property that should be used instead of the deprecated function. The existing functions still work in PeopleTools 8.0, however, they are being retained for backward compatibility only. New applications should be created using the new classes, methods, and properties.

Deprecated Function	***Use Instead***
ActiveRowCount	Rowset class, ActiveRowCount property
ClearSearchDefault	Field class, SearchDefault property
ClearSearchEdit	Field class, SearchEdit property
CompareLikeFields	Record class, CompareFields method
CopyFields	Record class, CopyFieldsTo method or CopyChangedFieldsTo method
CopyRow	Row class, CopyTo method
CurrEffDt	Rowset class, DeleteEnabled property
CurrEffRowNum	Row class, RowNumber property in combination with Rowset class, GetCurrEffRow method
CurrEffSeq	Rowset class, EffSeq property
CurrentRowNumber	Row class, RowNumber property
DeleteRecord	Record class, Delete method
DeleteRow	Rowset class, DeleteRow method
FetchValue	Field class, Value property
FieldChanged	Field class, IsChanged property
GetRelField	Field class, GetRelated method
GetStoredFormat	Field class, StoredFormat property
Gray	Field class, Enabled property
Hide	Field class, Visible property

Deprecated Function	***Use Instead***
HideRow	Row class, Visible property
HideScroll	Rowset class, HideAllRows method
InsertRow	Rowset class, InsertRow method
IsHidden	Row class, Visible property
NextEffDt	GetNextEffRow().REC.FIELD.Value
NextRelEffDt	GetNextEffRow.()REC.FIELD.GetRelated(rec.field).value
PriorEffDt	GetPriorEffRow().REC.Field.Value
PriorRelEffDt	GetPriorEffRow().REC.FIELD.GetRelated(rec.field).value
RecordChanged	Record class, IsChanged property
RecordDeleted	Record class, IsDeleted property
RecordNew	Row class, IsNew property
RemoteCall for Application Engine	CallAppEngine function
RowFlush	Rowset class, FlushRow method
RowScrollSelect	Rowset class, Select method
RowScrollSelectNew	Rowset class, SelectNew method
ScrollFlush	Rowset class, Flush method
ScrollSelect	Rowset class, Select method
ScrollSelectNew	Rowset class, SelectNew method

Deprecated Function	*Use Instead*
SetDefault	Field class, SearchDefault property
SetDefaultAll	Rowset class, SetDefault method
SetDefaultNext	GetNextEffRow().REC.FIELD.SetDefault()
SetDefaultNextRel	GetNextEffRow().REC.Field.GetRelated(REC.FIELD). SetDefault()
SetDefaultPrior	GetPriorEffRow().REC.FIELD.SetDefault()
SetDefaultPriorRel	GetPriorEffRow().REC.Field.GetRelated(REC.FIELD). SetDefault()
SetDisplayFormat	Field class, DisplayFormat property
SetLabel	Field class, Label property
SetSearchDefault	Field class SearchDefault property
SetSearchEdit	Field class, SearchEdit property
SetTracePC	If using API (Session object), use Trace Setting class properties.
SetTraceSQL	If using API (Session object), use Trace Setting class properties.
SortScroll	Rowset class, Sort method
TotalRowCount	Rowset class, RowCount property
Ungray	Field class, Enabled property
UnHide	Field class, Visible property
UnHideRow	Row class, Visible property

Deprecated Function	*Use Instead*
UnhideScroll	Rowset class, ShowAllRows method
UpdateValue	Field class, Value property

Deprecated Methods and Properties

The following are the methods and properties that have been deprecated.

Method or Property	*Use Instead*
Chart class, Refresh method	Chart class, SetData method Chart class, SetOLData method
Chart class, RotationAngle property	Chart class: XRotationAngle, YRotationAngle, and ZRotationAngle properties
Gantt class, Refresh method	Gantt class, SetTaskAppData method Gantt class, SetTaskData method
Query class, RunToTemplate method	No longer supported. This method remains in PeopleTools for PeopleSoft internal use only and should not be implemented in PeopleCode programs.
Session class, Connect method	%Session system variable
Session class FindTree, FindTreeStructure methods	GetTree, GetTreeStructure session class methods (respectively)
Tree collection	All tree collection methods and properties have been deprecated. Use GetTree to access a tree instead.
Tree structure collection	All tree structure collection methods and properties have been deprecated. Use GetTreeStructure to access a tree structure instead.
Item tree branch collection method, and the First, Last, and Next tree branch collection properties	NA

<table>
<tr><th>Method or Property</th><th>Use Instead</th></tr>
<tr><td>Tree class methods:
• FindNextDisplayObject
• FindPreviousDisplayObject</td><td>NA</td></tr>
<tr><td>Tree class properties:
• CheckLeafUserData
• DisplayLeaf
• DisplayNode
• IsReadOnly</td><td>NA</td></tr>
</table>

Deprecated XML Publisher Items

As of PeopleTools 8.50, a number of XML Publisher classes, methods and properties have been deprecated. These items remain in PeopleTools for backward compatibility only.

Deprecated XML Publisher Methods

<table>
<tr><th>Deprecated Method</th><th>Use Instead</th></tr>
<tr><td>ReportDefn class, SetRuntimeDataRowset method

Note. The rowset data source has been deprecated.</td><td>Convert the rowset to an XML file—for example, using the file layout capabilities of the File class is one way to accomplish this. Then, use the SetRuntimeDataXMLFile method of the ReportDefn class to process the XML file as a data source.</td></tr>
<tr><td>ReportDefn class, SetRuntimeDataXMLDoc method

Note. The XmlDoc data source has been deprecated.</td><td>Convert the XmlDoc object to an XML file—for example, using the genXmlFile method of the XmlDoc class is one way to accomplish. Then, use the SetRuntimeDataXMLFile method of the ReportDefn class to process the XML file as a data source.</td></tr>
</table>

Deprecated Method	Use Instead
Note. The rowset data source has been deprecated. RowsetDS class: • AddXMLData method • AddXSDSchema method • GetXMLData method • GetXSDSchema method • RowsetDS method	Convert the rowset to an XML file—for example, using the file layout capabilities of the File class is one way to accomplish this. Then, use the SetRuntimeDataXMLFile method of the ReportDefn class to process the XML file as a data source.

Deprecated XML Publisher Properties

Deprecated Property	Use Instead
OutputEditable	Use the Report Definition-Definition page (PSXPRPTPROP) to set report properties to override global properties.

XML Publisher Items No Longer Supported

As of PeopleTools 8.50, a number of XML Publisher classes, methods and properties are no longer supported. These items remain in PeopleTools for PeopleSoft internal use only and should not be implemented in PeopleCode programs.

- All methods and properties of the DataSourceDefn class.
- Certain methods and properties of the ReportDefn class.
- All methods and properties of the TemplateDefn class.
- All methods and properties of the TemplateFile class.
- All methods and properties of the TranslationFile class.
- All methods of the XMLPManager class.
- All methods of the QueryDS class.
- All methods and properties of the EFTPRocessor class.
- All methods and properties of the FOProcessor class.
- All methods and properties of the FormProcessor class.
- All methods and properties of the FOUtility class.
- All methods and properties of the PDFMapTool class.

- All methods and properties of the RTFProcessor class.

DataSourceDefn Class Methods

Method	***Use Instead***
DataSourceDefn*(DataSourceID,DataSourceType,Public)*	NA
Get()	NA
GetData*(&PromptRecord)*	NA
GetSampleData()	NA
GetSchema	NA

DataSourceDefn Class Properties

Property	***Use Instead***
ActiveFlag	NA
Description	NA
IsPublic	NA
LastUpdatedBy	NA
LastUpdateDTTM	NA
Name	NA
ObjectOwnerId	NA
RegisteredBy	NA
RegisteredDTTM	NA
Type	NA

ReportDefn Class Methods

Method	***Use Instead***
GetActiveTemplatesByDistributionChannel *(DistributionChannel,AsOfDate)*	NA
GetDataSource()	NA

Method	***Use Instead***
GetDefaultTemplate()	NA
GetTemplate*(TemplateID)*	NA
GetTemplateList()	NA
ProcessEtextReport*(TemplateID,LanguageCd,AsOfDate)*	ReportDefn class, ProcessReport method

ReportDefn Class Properties

Property	***Use Instead***
AllowRecipientEntry	NA
AllowViewerEntry	NA
ArchiveAfter	NA
BurstFieldName	NA
CategoryId	NA
ID	NA
IsReadOnly	NA
LastUpdateDTTM	NA
LastUpdatedOprId	NA
ObjectOwnerId	NA
RegisteredBy	NA
RegisteredDTTM	NA
SecurityFieldName	NA
SecurityIdType	NA
SecurityJoinTable	NA
TemplateControlFieldName	NA

TemplateDefn Class Methods

Method	*Use Instead*
TemplateDefn*(TemplateID)*	NA
Get()	NA
GetActiveTemplateFile*(AsOfDate)*	NA
GetTemplateFile()	NA
GetTemplateFileList()	NA

TemplateDefn Class Properties

Property	*Use Instead*
Description	NA
DistributionChannel	NA
ID	NA
IsReadOnly	NA
IsSubTemplate	NA
LanguageCode	NA
LastUpdateDTTM	NA
LastUpdatedOprId	NA
ObjectOwnerId	NA
RegisteredBy	NA
RegisteredDTTM	NA
ReportCategoryId	NA
Type	NA

TemplateFile Class Methods

Method	*Use Instead*
TemplateFile*(&TemplateDefn,EffDate)*	NA
GetFile()	NA

Method	***Use Instead***
GetMapFile()	NA
GetTranslationFile*(LanguageCD)*	NA
GetTranslationFileList()	NA

TemplateFile Class Properties

Property	***Use Instead***
EffectiveDate	NA
FileName	NA
MapFileName	NA
Status	NA

TranslationFile Class Method

Method	***Use Instead***
TranslationFile*(&TemplateDefn,EffectiveDate, LanguageCode)*	NA
GetFile()	NA

TranslationFile Class Properties

Property	***Use Instead***
Description	NA
FileName	NA
LanguageCode	NA
Status	NA

XMLPManager Class Methods

Method	*Use Instead*
GetDatasourceDefnList*(DataSourceId,DataSourceIdOp, DataSourceType,Owner,Descr,DescrOp,ObjectOwnerId, Active,CaseSensitive)*	NA
GetReportDefnList*(ReportID,ReportIdOp,Descr,DescrOp , TemplateId,TemplateIdOp,TemplateDescr, TemplateDescrOp,DatasourceType,DataSourceId, DataSourceIdOp,DataSourceOwner,TemplateType, ReportCategoryId,ReportStatus,CaseSensitive)*	NA
GetTemplateDefnList*(TemplateId,TemplateIdOp,Descr, DescrOp, TemplateType,DistChannel,ReportCategoryId, IsSubtemplate,CaseSensitive)*	NA

QueryDS Class Methods

Method	*Use Instead*
QueryDS()	NA
GetXMLData*(&Query,&Rowset,XSDLocation)*	NA
GetXMLSampleData*(&Query,&Rowset,Rows, XSDLocation*)	NA
GetXSDSchema*(&Query)*	NA

ETFProcessor Class Methods

Method	*Use Instead*
EFTProcessor()	
GenerateOutput*(EFTFile,XMLDataFile,OutputFile,Error)*	ReportDefn class, ProcessReport method
GenerateXSL*(RTFTemplateFile,XSLOutputFile,Error)*	ReportDefn class, ProcessReport method

EFTProcessor Class Properties

Property	*Use Instead*
ConfFile	ReportDefn class, ProcessReport method
ConfProp	ReportDefn class, ProcessReport method

FOProcessor Class Methods

Method	*Use Instead*
FOProcessor()	ReportDefn class, ProcessReport method
GenerateMergedOutput*(&XSLTemplateFileArray, &XMLDataFileArray,&XliffFileArray, OutputFileName, OutputFormat,Error)*	ReportDefn class, ProcessReport method
GenerateMergedOutputFO*(&FOFIleArray,OutputFile, OutputFormat,Error)*	ReportDefn class, ProcessReport method
GenerateOutput*(XSLTemplateFile,XMLDataFile, XLiffFile,OutputFile, OutputFormat,Error)*	ReportDefn class, ProcessReport method
GenerateOutputFO*(FOFileName,OutputFile, OutputFormat,Error*)	ReportDefn class, ProcessReport method

FOProcessor Class Properties

Property	*Use Instead*
ConfFile	ReportDefn class, ProcessReport method
ConfProp	ReportDefn class, ProcessReport method
Locale	ReportDefn class, ProcessReport method

FormProcessor Class Methods

Method	*Use Instead*
FormProcessor()	ReportDefn class, ProcessReport method
ExtractFieldNames*(PDFTemplateFileName, IncludeReservedNames,FieldNameArray,Error)*	ReportDefn class, ProcessReport method
ExtractFieldValues*(PDFTemplateFileName, IncludeReservedNames,FieldValueArray, Error)*	ReportDefn class, ProcessReport method
FillForm*(PDFTemplateFile,XMLDataFile, PDFOutputFile,Error*)	ReportDefn class, ProcessReport method

FormProcessor Class Properties

Property	*Use Instead*
ConfFile	ReportDefn class, ProcessReport method
ConfProp	ReportDefn class, ProcessReport method

Property	*Use Instead*
Locale	ReportDefn class, ProcessReport method

FOUtility Class Methods

Method	*Use Instead*
FOUtility()	ReportDefn class, ProcessReport method
GenerateFO*(XSLFile,XMLDataFile,OutputFOFile,Error)*	ReportDefn class, ProcessReport method
MergeFOs*(&FOFileArray,OutputFOFile,Error)*	ReportDefn class, ProcessReport method

FOUtility Class Property

Property	*Use Instead*
Prop	ReportDefn class, ProcessReport method

PDFMapTool Class Methods

Method	*Use Instead*
PDFMapTool()	ReportDefn class, ProcessReport method
EnableMap*(XSDFile,PDFTemplateFile,XMLInputFile, PDFOutputFile*)	ReportDefn class, ProcessReport method
extractMap*(PDFFile,OutMapFile)*	ReportDefn class, ProcessReport method
processForm*(PDFTemplateFile,XMLDataFile,MapFile, OutputPDFFile*)	ReportDefn class, ProcessReport method

PDFMapTool Class Property

Property	*Use Instead*
Locale	ReportDefn class, ProcessReport method

RTFProcessor Class Methods

Method	*Use Instead*
RTFProcessor()	ReportDefn class, ProcessReport method

Method	*Use Instead*
GenerateXSL*(RTFTemplateFile,XSLOutputFile,Error)*	ReportDefn class, ProcessReport method
GenerateXSLXliff(*RTFTemplateName,XSLOutputFile, XliffOutputFile,Error*)	ReportDefn class, ProcessReport method

RTFProcessor Class Properties

Property	*Use Instead*
ConfFile	ReportDefn class, ProcessReport method
ConfProp	ReportDefn class, ProcessReport method
Locale	ReportDefn class, ProcessReport method

Deprecated Messaging PeopleCode Functions, Methods and Properties

In PeopleTools 8.48, messaging changed significantly. Many of the messaging PeopleCode functions, methods and properties are either deprecated or no longer supported, or their syntax changed.

See Chapter 25, "Message Classes," page 1245.

See *PeopleTools 8.51 PeopleBook: PeopleSoft Integration Broker,* "Understanding PeopleSoft Integration Broker."

Changed Messaging PeopleCode

The following are still valid, however, their syntax changed significantly.

- CreateMessage built-in function
- IntBroker class, GetMessageErrors method
- IntBroker class, SetMessageError method

Deprecated Messaging PeopleCode Built-in Functions

The following table lists the deprecated messaging PeopleCode built-in functions, as well as the method that should be used instead.

Deprecated Function	*Replacement Method*
CancelPubHeaderXmlDoc	IntBroker class method Cancel, specifying a message type of %IntBroker_BRK
CancelPubXmlDoc	IntBroker class method Cancel, specifying a message type of %IntBroker_PUB

Deprecated Function	*Replacement Method*
CancelSubXmlDoc	IntBroker class method Cancel, specifying a message type of %IntBroker_SUB
ConnectorRequest	IntBroker class method ConnectorRequest
ConnectorRequestURL	IntBroker class method ConnectorRequestURL
GetMessage	IntBroker class method GetMessage
GetMessageXmlDoc	Message class method GetXmlDoc
GetPubHeaderXmlDoc	IntBroker class method GetMessage specifying a message type of %IntBroker_BRK
GetPubXmlDoc	IntBroker class method GetMessage specifying a message type of %IntBroker_PUB
GetSubXmlDoc	IntBroker class method GetMessage specifying a message type of %IntBroker_SUB
GetSyncLogData	IntBroker class method GetSyncLogData
InBoundPublishXmlDoc	IntBroker class method InboundPublish
IsMessageActive	IntBroker class method IsOperationActive
PublishXmlDoc	IntBroker class method Publish
ReSubmitPubHeaderXmlDoc	IntBroker class method Resubmit specifying a message type of %IntBroker_BRK
ReSubmitPubXmlDoc	IntBroker class method Resubmit specifying a message type of %IntBroker_PUB
ReSubmitSubXmlDoc	IntBroker class method Resubmit specifying a message type of %IntBroker_SUB
SetChannelStatue	IntBroker class method SetQueueStatus
SyncRequestXmlDoc	IntBroker class method SyncRequest
UpdateXmlDoc	IntBroker class method UpdateXmlDoc

The following built-in functions are no longer supported. You will receive an error if you try to use them in a PeopleCode program.

- CreateWSDLMessage
- GetArchPubHeaderXmlDoc
- GetArchPubXmlDoc

- GetArchSubXmlDoc
- GetArchSubXmlDoc
- GetArchSyncLogData
- GetSyncLogData

Note. ReturnToServer is a special case of a built-in function that's no longer supported. The deprecated handler for OnRequest subscriptions cannot be upgraded. ReturnToServer can only be used in an OnRequest event fired using the deprecated handler. This means that ReturnToServer no longer works and is not valid in any case other than when the code has already been written and used in a deprecated handler.

Deprecated Message Class Methods and Properties

The following are the Message class methods and properties that have been deprecated.

Deprecated Method	***Replacement Method***
GetQueryString	IBConnectorInfo collection GetQueryStringArgName method
InBoundPublish	IntBroker class InBoundPublish method
Publish	IntBroker class Publish method
SetQueryString	IBConnectorInfo collection AddQueryStringArg method
SetStatus	IntBroker class SetStatus method
SyncRequest	IntBroker class SyncRequest method
Update	IntBroker class Update method

The following are the Message class properties that have been deprecated.

Deprecated Property	***Replacement Property***
ChannelName	Message class QueueName property
Cookies	IBConnectorInfo collection Cookies property
DefaultMessageVersion	Message class OperationVersion property
GUID	Message class TransactionId property
IsActive	Message class IsOperationActive property
PubId	Message class QueueSeqId property
SubName	Message class ActionName property

Deprecated Property	*Replacement Property*
SubscriptionProcessId	Message class TransactionId property
MessageChannel	IBInfo class MessageQueue property
MessageType	IBInfo class OperationType property

The following Message class properties are no longer supported. You will receive an error if you try to use them in your PeopleCode program.

- MessageDetail
- PublicationId

Additional Messaging Deprecated Methods and Properties

The following additional methods and properties are either deprecated or no longer supported:

- The IBInfo class method LoadConnectorPropFromTrx is no longer supported.
- The IBInfo class MessageChannel property has been deprecated. Use the IBInfo class MessageQueue property instead.
- The IBInfo class PublicationID property is no longer supported.

Integration Broker Web Service Discovery Class No Longer Supported

The Integration Broker Web Service Discovery class is no longer supported. The following methods are empty and should not be used.

- FindCIWebServices
- FindMsgWebServices
- FindWebServices
- GetCIWebServiceWSDL
- GetMsgWebServiceWSDL
- GetWSDL

Functions No Longer Supported

The following functions no longer work in release 8.4 and beyond.

Function	*Description*
%MessageAgent	Used with Message Agent activities

Function	***Description***
ChDir	Used only on the client with DOS
ChDrive	Used only on the client with DOS
CheckMenuItem	Used with menu items
Codeb	Used before true Unicode integration
DBCSTrim	Used before true Unicode integration
Findb	Used before true Unicode integration
GetControl	Used with ActiveX controls
GetControlOccurence	Used with ActiveX controls
GetCwd	Used only on the client with DOS
GetMessageInstance	Used with messaging
GetPubContractInstance	Used with messaging
GetSelectedTreeNode	Used with dynamic trees
GetSubContractInstance	Used with messaging
GetTreeNodeParent	Used with dynamic trees
GetTreeNodeRecordName	Used with dynamic trees
Lenb	Used before true Unicode integration
RefreshTree	Used with dynamic trees
ScheduleProcess	Used with PeopleSoft Process Scheduler

Function	***Description***
Substringb	Used before true Unicode integration
UnCheckMenuItem	Used with menu items
WinEscape	Used only on client
WinExec	Used only on client

Index

Symbols

%Menu conditions 612
%Request system variable 1059
%Response system variable 1059
%SyncServer system variable 2263, 2271
%ThisMobileObject system variable 1389
1159Separator property 2202
2359Separator property 2203

A

AbnContentProvider property
Folder class 1664
AbnDataSource property
Folder class 1665
AbnPeopleCode property
Folder class 1665
AbsoluteContentURL property
Content Reference class 1685
ContentReference links 1723
AbsolutePortalURL property
Content Reference class 1685
ContentReference links 1724
abstract
declaring methods/properties 202
understanding base classes 223
AcceptedTaskList property
AgentPhysQueueTasks class 2442
PhysicalQueue class 2449
actiondttm property 1476
ActionName property 1295
ActivateModel method 1565
ActivateObjective method 1566
ActiveNode property 1646
ActiveRowCount property 2161
AddAEAttribute method
IBInfo class 1336
AddAllFields method 1961
AddArgument method 147
AddAttachment method
IBInfo class 1338
MCFOutboundEmail class 1178
AddAttribute method
IBInfo class 1339
Incoming Business Interlink class 357
XmlNode class 2553
AddAttributeNS method 2554
AddBody method 2219
AddBodyPart method 1175
addCategory method
FeedDoc class 903
FeedEntry class 919
AddCDataSection method 2555
AddComment method
Incoming Business Interlink class 358
XmlNode class 2557
AddConnectorProperties method 1374
AddContainerAttribute method
IBInfo class 1340
addContributor method
FeedEntry class 919
AddCriteria method 1962
AddCubeCollection method 59
AddCube method 59
AddDimension method 60
AddDistributionOption method
PostReport class 1827
ProcessRequest class 1846
AddDoc method 311
AddDropDownItem method 937
AddElement method 2558
AddElementNS method 2559
addEnclosure method
FeedEntry class 920
AddEntityReference method 2560
addEntry method
FeedDoc class 904
AddEnvelope method 2220
AddExplicitDimensionSet method 61
AddExpr1Expression method 1998
AddExpr1Field method 1999
AddExpr2Expression method 2000
AddExpr2Field1 method 2000
AddExpr2Field2 method 2001
AddExpr2List method 2002
AddExpr2Subquery method 2002
AddExpression method 1963
AddFault method 2222
AddHavingCriteria method 1963
AddHeader method
MCFBodyPart class 1117
MCFOutboundEmail class 1179
SOAPDoc class 2224
adding
RuleExpression 129, 142
AddInputRow method 313
AddListValue method 2023
AddMember method 26
Add method 2320
AddMethod method 2225
AddModel method 173
AddNextDoc method 314
AddNode method 2561
AddNotifyInfo method
ProcessRequest class 1847
AddOrganizer method 62
addParameter method
DataSource class 816
AddParm method 2227
AddProcessInstruction method
Incoming Business Interlink class 359
XmlNode class 2562
AddPrompt method 1922
AddQueryOutputField method 1964
AddQueryRecord method 1965
AddQuerySelectedField method 1965
AddQuerySelect method 1923
AddQueryStringArg method 1375
AddRecord method 174
AddReference method 2271
AddReferenceType method 2272

AddRuleExpression method
ExpressionBlock class 142
RuleDefn class 129
AddSearchFieldCriteria method 2619
AddStep method 5
AddText method
Incoming Business Interlink class 361
XmlNode class 2563
Add To CLASSPATH parameter
psappsrv.cfg 1086
AddTrackingURL method 1924
AddTranslateExpression method 1984
AddTranslateField method 1985
AddUnion method 1956
AddUserFunction method 62
AddUser method 1235
addUserValue method
DataSourceParameter class 832
AddValue method 317
AdminPersonalizationPage property
DataSource class 826
AdvancedSearchQueries method 1905
AdvancedSearchRecords method 1908
AE_ID property
SCHED_INFO class 706
AESection
class *See Also* AESection class
objects *See Also* AESection objects
AESection class
accessing 2
example 3
list of functions, methods, properties, returns 2650
methods *See Also* AESection class methods
properties
See Also AESection class properties
understanding 1
AESection class methods
AddStep 5
Close 6
Open 7
Save 8
SetSQL 9
SetTemplate 10
AESection class properties 11
AESection objects
declaring/scope 5
deleting steps 3
opening/getting 2
Agent class
list of methods, properties, returns 2774
methods 2434
properties *See Also* Agent class properties
understanding 2433
Agent class properties
AgentID 2436
AgentProps 2436
AgentTasks 2437
Buddy 2438
Language 2438
Name 2438
NickName 2438
PhysicalQueueID 2439
TotalPhysicalQueues 2439
Agent constructor 2425
AgentID property
Agent class 2436
AgentPhysQueueTasks class 2443
Task class 2457
TaskList class 2460
AgentID property
AgentPhysQueueProps class 2440
AgentPhysQueueProps class
list of properties, returns 2775
properties 2440
understanding 2439
AgentPhysQueueProps constructor 2426
AgentPhysQueueTasks class
list of methods, properties, returns 2775
methods 2441
properties 2442
understanding 2441
AgentPhysQueueTasks constructor 2426
Agent property 2449
AgentProps property 2436
agents
Agent class *See Also* Agent class
Agent constructor 2425
AgentPhysQueueProps class
See Also AgentPhysQueueProps class, AgentPhysQueueTasks class
Agent property 2449
Mobile Agent *See Also* Mobile Agent
AgentTasks property 2437
aggregate
calculating 106
dimension 101
getting persistent 114
mappings 110, 115
record name 122
sequence 94
setting 104
setting persistent 119
user function 94
Aggregate property
QueryExpression class 2019
QueryField class 1986
AggregateRecName property 122
AggregateSequence property 94
AggregationUserFunction property 94
algorithms
accessing PortalRegistry objects 1592
ConflictAlgorithm property 2277
merge algorithm 2266
using for conflicts 2266
AliasName property 1295
AllChildCount property 2346
AllChildNodeCount property 2346
AllowAnyJoin property 2038
AllowChangesToRequired property
DataSourceParameter class 836
AllowCustomParameters property
DataSource class 826
AllowDistinct property 2038
AllowExpressions property 2038
AllowHTTPS property 2515
AllowMoreEntries property
FeedDoc class 910
AllowRealTimeFeedSecurity property
DataSource class 827
AllowSubqueries property 2038
AllowUnions property 2038
AllValuesAudit property 2324
AllValues property 2385
analytic calculation engine
running asynchronously 14

analytic calculation engine classes
    built-in functions 16
    calling built-in functions 143
    creating expression blocks 141
    data types 16
    error handling 15
    establishing hierarchies 14
    referencing members 149
    scope 16
Analytic Calculation Engine Metadata classes
    constructor 57
    creating 57
    data types 55
    error handling 55
    importing 56
    understanding 53
    using 54
analytic grid
    error handling 162
    populating 165
AnalyticGrid class
    data type 162
    GetColumn method 163
    GetCubeCollection method 164
    LoadData method 165
    methods 163
    properties 167
    SetAnalyticInstance method 165
AnalyticGrid class properties
    Inactive 167
    Label 167
    ShowGridLines 167
    SlicerVisible 168
    SummaryText 168
analytic grid columns
    specifying names 163
analytic grids 160
    activating 167
    displaying grid lines 167
    displaying slicer 168
    labels 167
    scope 162
    secondary pages 160
    setting summary text 168
AnalyticInstance class
    methods 16
    properties 25
analytic instances
    analytic type definition name 25
    asynchronous status 16
    class properties 25
    copying 18
    creating 1491
    deleting 19, 1494, 1511
    determining presence in optimization metadata 1517
    getting ACE model 20
    getting list of 1514
    ID property 25
    inserting 1515
    loading 21
    loading into analytic servers 1491
    loading into optimization engines 1491
    messages 26
    running asynchronously 22
    running synchronously 23
    status 17
    terminating 24
    unloading 24
analytic model
    accessing
        trees 33
        cubes 74
        cube collections 75
        cube collection names 76
        cube names 77
        dimensions 77
        dimension names 78
        explicit dimension sets 79
        explicit dimension set names 80
        organizers 80
        organizer names 81
        user functions 82
        user function names 82
        value changes 91
        iterations 92
        messages 92
    adding
        dimension members 26
        cubes 59
        cube collections 59
        dimensions 60
        explicit dimension sets 61
        organizer 62
        user functions 62
    attaching trees 27
    calculating cubes 28
    cell properties 30
    checking validity 91
    class methods 59
    class properties 90
    copying
        cubes 63
        cube collections 64
        dimensions 65
        explicit dimension sets 65
        model definitions 66
        user functions 67
    creating 68, 155
    deleting
        models 68
        cubes 69
        cube collections 70
        dimensions 71
        explicit dimension sets 71
        organizers 72
        user functions 73
    describing 91
    detaching trees 29
    getting 74
    getting a reference to 20
    getting CubeCollection objects 30
    getting members 31
    getting the definition 58
    messages property 36
    metadata classes example 155
    naming 93
    recalculating loaded model 34
    renaming
        definitions 83
        cubes 84
        cube collections 85
        dimensions 85
        explicit dimension sets 86
        organizers 87
        user functions 88

resolving circular dependencies 93
saving 89
setting circular formula warnings 90
validating 89
AnalyticModel class
methods 26
property 35
AnalyticModelDefn class
constructor 58
creating instance of 58
methods 59
properties 90
analytic servers
loading analytic instances 1491
analytic type classes
constructors 172
creating 171
error handling 170
example 188
getting the definition 172
importing 171
using 170
analytic type definitions
adding models 173
adding records 174
class properties 182
copying 175
creating 174
deleting 176
deleting analytic instances 1494
deleting models 176
deleting records 177
GetModel method 178
GetModelNames method 179
GetRecord method 180
GetRecordNames method 180
instantiating 174, 178
name of 25
renaming 181
saving 182
using callback records 1544
AnalyticTypeDefn class
constructor 172
methods 173
AnalyticTypeDefn class properties
AppClassPath property 182
Comments property 182
Description 183
Name 183
OwnerID 183
AnalyticTypeModelDefn class properties 183
Name 183
Type 184
analytic type objects
data types 170
scope 171
AnalyticType property 25
AnalyticTypeRecordDefn class
methods 184
properties 186
AOL instant messaging 1233
API instantiation methods 2186
API property 2204
API Repository
accessing via PeopleCode 235
accessing via Visual Basic 239
understanding 233
API Repository classes
Bindings *See Also* Bindings class
ClassInfo *See Also* ClassInfo class
list of methods, properties, returns 2793
MethodInfo *See Also* MethodInfo class
Namespaces *See Also* Namespaces class
PropertyInfo *See Also* PropertyInfo class
Repository *See Also* Repository class
API Repository collections
Bindings collection
See Also Bindings collection
ClassInfo collection
See Also ClassInfo collection
MethodInfo collection
See Also MethodInfo collection
Namespaces collection
See Also Namespaces collection
PropertyInfo collection
See Also PropertyInfo collection
AppClassPath property 182
AppendItem method
Collection class 753
application classes
Agent class *See Also* Agent class
AgentPhysQueueProps class
See Also AgentPhysQueueProps class
AgentPhysQueueTasks class
See Also AgentPhysQueueTasks class
BPEL *See Also* BPEL classes
Broadcast class *See Also* Broadcast class
calling via Java 1094
declaring 224
declaring abstract methods/properties 202
declaring constants *See Also* constants
declaring external functions 206
declaring external interfaces 193
declaring private instance-variables/methods 198
declaring protected properteis and methods 194
defining methods 199
designing base classes 223
downcasting 219
extending 192
functions *See Also* application class functions
functions, returns 2666
global variables 217
handling exceptions *See Also* Exception class
importing names 209, 224
instance variables 217
language constructs
See Also application class language constructs
LogicalQueue class
See Also LogicalQueue class
mail classes *See Also* mail classes
MCFFactory class
See Also MCFFactory class
naming 191, 193, 208
notification *See Also* Notification classes
overriding properties 214
PhysicalQueue class
See Also PhysicalQueue class
placing variable declarations 217
PrcsApi class 1880
referencing superclasses 225
scope 225
self-referencing 210
structure 190

superclass referencing 218
Task class *See Also* Task class
TaskList class *See Also* TaskList class
understanding 189
understanding collections
See Also PeopleCode collections
understanding comments *See Also* comments
universal queue classes
See Also universal queue classes
using 190
using properties 212
using variables *See Also* variables
Util class *See Also* Util class
utility classes 224
application class functions
Import 228
Method 230
application class language constructs
Class 225
Get 227
Interface 229
Set 231
ApplicationData property 2457
Application Engine
See Application Engine, Application Engine
File Layout example 1026
logging errors 2181
programs
See Also Application Engine programs
running optimization transactions 1493
running PeopleCode 1492
sections *See Also* AESection class
using SQL objects 2247
using with Business Interlink 310
Application Engine handler
IBInfo class 1336
Application Engine programs
editing OPT_CALL 1506
editing PT_OPTCALL 1502
terminating 1490
using optimization PeopleCode 1490
application packages *See Also* application classes
naming 208
PT_BPEL 288
PT_MCF_MAIL 1101
PT_WF_NOTIFICATION 1446
PT_WF_WORKLIST 1446
using for classes 189
ApplicationRelease property
FeedDoc class 910
application servers
managing 1088
running optimization transactions 1493
running PeopleCode 1491
using optimization PeopleCode 1489
Approved property 1946
ApproveDtTm property 1947
ApproveOprId property 1946
ApprovePrivateQuery property 2039
ApprovePublicQuery property 2039
ApproveUserId property 1946
AppServerDomain property 1362
AppsRelease property 1646
arguments
adding 147
get method 148
Arguments property 252
Array class
creating 257
functions, methods, properties, returns 2667
methods *See Also* Array class methods
properties 284
understanding 257
Array class methods
Clone 265
Find 266
Get 267
Join 268
Next 271
Pop 272
Push 273
Replace 275
Reverse 277
Set 278
Shift 278
Sort 279
Subarray 282
Substitute 282
Unshift 284
Array class properties
Dimension 285
Len 285
array objects
creating/populating multi-dimensional 261
creating empty 260
declaring 264
flattening and promoting 264
populating 259
removing items 260
scope of 265
arrays
Array class *See Also* Array class
copying between PeopleCode and Java 1097
Java 1097
removing elements 275
understanding *See Also* array class
AssignedPagelet
class *See Also* AssignedPagelet class
collection
See Also AssignedPagelet collection
Content Reference class AssignedPagelets
property 1686
TabDefinition class AssignedPagelets
property 1738
AssignedPagelet class
list of, properties, returns 2827
properties
See Also AssignedPagelet class properties
understanding 1749
AssignedPagelet class properties
Column 1749
LayoutBehavior 1749
PageletName 1750
Row 1750
AssignedPagelet collection
list of methods, properties, returns 2827
methods
See Also AssignedPagelet collection
methods
properties 1754
understanding 1750
AssignedPagelet collection methods
DeleteItem 1751
First 1751
InsertItem 1752
ItemByName 1753

Next 1754
AssignedPagelets property
Content Reference class 1686
TabDefinition class 1738
AssignedTaskList property
AgentPhysQueueTasks class 2442
PhysicalQueue class 2449
assignment class
creating 132
Expression property 133
GenerateRule method 132
method 132
properties 133
Variable property 133
AsyncFFSend class
method 291
understanding 291
AsyncFFSend class method
OnRequestSend 291
AsyncFFSend constructor 289
asynchronous mode
loading analytic instances 1491
running optimization transactions 1492
running transactions 1499
understanding the RunAsynch method 1534
AttachCube method 108
AttachDimension method
CubeDefn class 98
ExplicitDimensionSet class 95
AttachList property 1152
ATTACHMENT_URL property
Utility class 875
AttachmentRoot property 1143
attachments
IBInfo class
1338, 1342, 1344, 1348, 1349, 1360
IsAttachment property 2504
number of 1368
ResponseAsAttachment property 1370
using events 2269
AttachmentURL property 1123
AttachPart method 125
AttachSizes property 1152
AttachTree methods 27
Attribute collection
methods
See Also Attribute collection methods
properties 1710
understanding 1707
Attribute collection methods
DeleteItem 1707
First 1708
InsertItem 1708
ItemByName 1709
Next 1710
Attribute collection properties 1710
AttributeCount property 368
attributes
Attribute collection
See Also Attribute collection
AttributeCount property 368
AttributeValue class 1704
Content Reference class Attributes property
1686
Folder class Attributes property 1666
GetAttributeName method 2568
GetAttributeValue method 2569
GetPeerDefaultAttributesByName method
1404
Incoming Business Interlink class
AddAttribute method 357
Incoming Business Interlink class
GetAttributeName method 364, 365
PageletCategory class Attributes property
1768
Pagelet class Attributes property 1778
PeerDefaultAttributes class 1439
TabDefinition class Attributes property 1739
understanding PortalRegistry 1590
using 1819
AttributesCount property 2589
Attributes property
Content Reference class 1686
ContentReference links 1724
Folder class 1666
PageletCategory class 1768
Pagelet class 1778
TabDefinition class 1739
AttributeValue class
list of properties, returns 2820
properties
See Also AttributeValue class properties
understanding 1704
AttributeValue class properties
Label 1705
Name 1705
Translatable 1706
Value 1706
AttributeValue collection 2821
attrName property 2626
attrValue property 2626
AuditByName method 2357
AuditDetails property 2386
Audit method 2356
authentication
AuthTokenDomain property 1066
AuthType property 1067
Servlet property 1074
signing on to PeopleSoft 2180
AuthorAccess property
Content Reference class 1687
ContentReference links 1724
Folder class 1666
PageletCategory class 1769
Pagelet class 1779
TabDefinition class 1739
understanding 1591
Authorized property
Content Reference class 1687
ContentReference links 1725
Feed class 789
Folder class 1667
PageletCategory class 1769
Pagelet class 1779
AuthorizeEmailAttach method 1157
Author property
Content Reference class 1686
ContentReference links 1724
FeedEntry class 924
Folder class 1666
PageletCategory class 1769
Pagelet class 1778
TabDefinition class 1739
AuthTokenDomain property 1066
AUTHTYPE_PERM property
Utility class 875

AUTHTYPE_ROLE property
Utility class 875
AuthType property 1067
AvailableCategories property 1738
AvailableCategory
class *See Also* AvailableCategory class
collection
See Also AvailableCategory collection
AvailableCategory class
list of properties, returns 2828
properties 1755
understanding 1755
AvailableCategory collection
list of methods, properties, returns 2828
methods
See Also AvailableCategory collection methods
properties 1758
understanding 1756
AvailableCategory collection methods
First 1756
ItemByName 1757
Next 1757
AvailablePagelet
class *See Also* AvailablePagelet class
collection
See Also AvailablePagelet collection
objects 1758
AvailablePagelet class
list of, properties, returns 2828
properties
See Also AvailablePagelet class properties
understanding 1758
AvailablePagelet class properties
CategoryLabel 1759
CategoryName 1759
Column 1759
LayoutBehavior 1759
PageletLabel 1760
PageletName 1760
Row 1760
AvailablePagelet collection
list of methods, properties, returns 2829
methods
See Also AvailablePagelet collection methods
properties 1762
understanding 1760
AvailablePagelet collection methods
First 1761
ItemByName 1761
Next 1762
AvailablePagelets property
AvailableCategory class 1755
TabDefinition class 1739
AvgExecTime property 2036
AvgFetchTime property 2036
AvgNumRows property 2037
AxisEndDateTime property 506

B

BackgroundFile property 2628
BackupServer property 1208
BackupSMTPPort property 1187
BackupSMTPServer property 1187
BackupSMTPSSLClientCertAlias property 1187
BackupSMTPSSLPort property 1187
BackupSMTPUserName property 1188
BackupSMTPUserPassword property 1188
BackupSMTPUseSSL property 1188
BackupSSLClientCertAlias property 1209
BackupSSLPort property 1209
BackupUserName property 1209
BackupUserPassword property 1210
BackupUseSSL property 1210
badaddresses property
MCFMailUtil class 1173
base classes
adding members 219
designing 223
MCFFactory class
See Also MCFFactory class
BCC property
MCFEmail class 1128
MCFOutboundEmail class 1189
BIDoc class 2873
BIDocs 306
BindFlag property 2019
Bindings
collection *See Also* Bindings collection
methods 243
property 241
Bindings class
list of properties, returns 2793
properties 243
Bindings collection
list of properties, returns 2793
methods 242
properties 242
Bindings collection methods 242
Bindings collection properties 242
Bindings methods 243
Bindings properties 243
Bindings property 241
BodyNode property 2231
BounceTo property
MCFEmail class 1128
MCFOutboundEmail class 1189
BoxMaxDisplayItems property 547
BPEL
classes *See Also* BPEL classes
BPEL classes 2669
AsyncFFSend *See Also* AsyncFFSend class
BPELUtil *See Also* BPELUtil class
constructors
See Also BPEL class constructors
creating BPEL objects 288
declaring data types 288
IBUtil *See Also* IBUtil class
importing 288
scope 287
understanding 287
BPEL classes constructors
AsyncFFSend 289
BPELUtil 290
IBUtil 290
BPEL objects, creating 288
BPELUtil class
LaunchASyncBPELProcess 295
LaunchSyncBPELProcess 296
methods 292
understanding 292
UpdateConnectorResponseProperties 297

BPELUtil class methods
GetASyncBPELProcessInstanceUrl 292
GetBPELProcessBrowserUrl 293
GetOperationType 293
GetSyncBPELProcessInstanceUrl 294
BPELUtil constructor 290
Branch collection
list of properties, returns 2857
property 2302
understanding 2302
Branch collection property 2302
branches
Branches property 2386
BranchImageName property 2386
Branch method 2326
collection *See Also* Branch collection
Branches property 2386
BranchImageName property 2386
BranchLevel property 2386
Branch method 2326
BranchName property 2387
Broadcast class
method 2443
Broadcast constructor 2427
Broadcast method 2444
BrowserPlatform property 1067
browsers
BrowserPlatform property 1067
BrowserType property 1067
BrowserURL property 2450
BrowserVersion property 1068
changing portal templates 1639
using plug-ins for iScripts 1056
BrowserType property 1067
BrowserURL property 2450
BrowserVersion property 1068
Buddy property 2438
buffers
component 1248
Record class 2077
RowsetCache class 2171
supporting batch input/output 305
supporting dynamic output 305
using Business Interlink objects 304
BuildSearchIndex method 2474
BuildType property 2513
built-in functions *See* PeopleCode functions
deprecated 2895
document classes 730
BulkExecute method 319
BulkMode property 2254
busactivity property 1476
buseventname property 1476
Business Interlink
class *See Also* Business Interlink class
objects *See Also* Business Interlink objects
Business Interlink class
configuring parameters 372, 373, 374
incoming
See Also Incoming Business Interlink class
methods *See Also* Business Interlink methods
properties 372
understanding 303
using with Application Engine 310
Business Interlink class methods
AddDoc 311
AddInputRow 313
AddNextDoc 314
AddValue 317
BulkExecute 319
choosing 304
Clear 324
Execute 325
executing Business Interlink objects 305
FetchIntoRecord 327
FetchIntoRowset 329
FetchNextRow 332
GetCount 334
GetDoc 335
GetFieldCount 338
GetFieldType 339
GetFieldValue 340
GetInputDocs 341
GetNextDoc 342
GetOutputDocs 344
GetPreviousDoc 345
GetStatus 347
GetValue 349
InputRowset 350
MoveFirst 352
MoveNext 353
MoveToDoc 354
ResetCursor 356
supporting batch input/output 305
supporting dynamic output 305
supporting rowsets 305
using flat table methods 305
using hierarchical data (BIDocs) 306
Business Interlink objects
declaring/scope 311
declaring as global 310
executing 305
input structures 306
output structures 308
supporting batch input/output 305
supporting dynamic output 305
supporting rowsets 305
understanding 303
understanding states 310
using 304
using flat table methods 305
using hierarchical data 306
Business Interlink plug-in configuration 373
Business Interlinks class
list of deprecated functions, methods, properties, returns 2872
busprocname property 1476
ByPassSignOn property 1067

C

CalcAggregates property 106
CalculateCube method 28
callback
OptBase callback methods 1543
callback property 186
callback records
designating in analytic type definitions 1544
using Optbase Callback methods 1543
Cancel method 615
IntBroker class 1312
CanCreatePublic property 2039
SEC_PROFILE class 709
CanCreateWorkFlow property 2039

CanModify property
    SEC_PROFILE class 709
CanModifyQuery property 2040
CanRunQuery property 2040
    SEC_PROFILE class 709
CanRunToCrystal property 2040
CanRunToExcel property 2040
CascadedPermissions property
    Content Reference class 1687
    ContentReference links 1725
    Folder class 1667
    PageletCategory class 1769
    Pagelet class 1779
    understanding 1593
CascadedRolePermissions property
    Content Reference class 1688
    ContentReference links 1726
    Folder class 1668
Cascade property 1593, 1711
cascading 1592
Categories property
    FeedDoc class 910
    FeedEntry class 925
CategoryID property
    Feed class 789
CategoryLabel property 1759
CategoryName property
    AvailableCategory class 1755
    AvailablePagelet class 1759
    SelectedPagelet class 1797
Category property 2387
causes 100
CC property
    MCFEmail class 1128
    MCFOutboundEmail class 1189
CenterFocusedNode property 526
CGI variables
    AUTH_TYPE 1067
    REMOTE_USER 1072
    SCRIPT_NAME 1074
ChangeOnInit property 2162
ChannelName property 1295
Channel property 1459
characters
    Charset property 1082
    subscribing to character fields 1251
    using character values 973
Charset property 1082, 1124
    MCFOutboundEmail class 1189
Chart class
    chart types 396
    handling errors 397
    list of methods 2670
    list of properties 2671
    methods *See Also* Chart class methods
    objects 397
    properties *See Also* Chart class properties
    selecting color values 390, 434
    terminology
        general 395
Chart class appearance properties
    Height 456
    LineType 460
Chart class axes methods
    SetXAxisLabels 452
    SetYAxisLabels 452
Chart class axes properties
    YAxisTicks 475
Chart class chart properties
    XAxisScaleResolution 469
Chart class color methods
    SetDataColor 438
Chart class data methods
    SetDataHints method 439
    SetDataSeries method 443
    SetDataURLs method 443
    SetDataXAxis method 444
    SetDataYAxis method 446
Chart class data properties
    DataWidth 454
Chart class legend methods 446
Chart class methods
    Refresh method 432
    SetColorArray 433
    SetDataAnnotations method 437
    SetData method 435
Chart class overlay methods
    SetOLData 447
    SetOLDataSeries 450
    SetOLDataXAxis 450
    SetOLDataYAxis 451
Chart class properties
    DataStartRow 454
    GridLines 455
    HasLegend 456
    ImageMap 457
    IsDrillable 457
    IsPlainImage 458
    IsYAxisInteger 458
    LegendMaxEntries 459
    LegendPosition 459
    LegendStyle 460
    MainTitle 461
    MainTitleOrient 461
    MainTitleStyle 461
    OLLineType 462
    OLType 462
    RevertToPre850 463
    RotationAngle 463
    Style 464
    StyleSheet 464
    Type 465
    Width 466
    XAxisCross 466
    XAxisCrossPoint 467
    XAxisLabelOrient 468
    XAxisMax 468
    XAxisMin 468
    XAxisPrecision 469
    XAxisStyle 470
    XAxisTicks 470
    XAxisTitle 470
    XAxisTitleOrient 471
    XAxisTitleStyle 471
    XRotationAngle 471
    YAxisCrossPoint 472
    YAxisLabelOrient 472
    YAxisMax 473
    YAxisMin 473
    YAxisPrecision 473
    YAxisScaleResolution 474
    YAxisStyle 474
    YAxisTitle 475
    YAxisTitleOrient 475
    YAxisTitleStyle 476
    YRotationAngle 476

ZRotationAngle 476
Chart class Scatter chart property
ShowCrossHair 464
Chart class Scatter methods
SetDataGlyphScale 438
SetOLDataAnnotations 448
SetOLDataGlyphScale 449
charting
Chart class
chart types 396
component processor 383
creating charts via iScripts 596
creating Gantt charts 585
creating organization charts 588
creating rating box charts 592
examples 557
label concerns 383
rating box charts 425
terminology
Gantt chart 400
Organization Chart 407
translating values 383
using style sheets 383
charting class
lists of functions, methods, properties, and returns 2670
charting classes
creating a charting using an iScript 382
creating a chart on a page 379
creating charts 378
understanding 375
Charting classes
Gantt class methods
See Also Gantt class methods
chart objects 397
charts
charting classes *See Also* charting classes
creating Gantt charts 585
creating organization charts 588
creating rating box charts 592
creating via iScripts 596
types 396
ChartScrollType property 526
chat 1233
CheckAll method 1235
CheckAsyncStatus method 16
CheckForChanges method 2273
CheckOptEngineStatus method 1518
CheckQryForTreePrompt method
UTILITY class 710
CheckQrySecurity method
UTILITY class 711
CheckStatus method 17
ChildCount property 2119
ChildLeafCount property 2347
ChildNodeCount property
Incoming Business Interlink class 369
Node class 2347
XmlNode class 2589
chunking, about 1951
CI collection
list of methods, properties, returns 2877
methods 1424
properties 1424
understanding 1394, 1423
circular dependencies 101
CircularFormulaWarn property 90
classes
custome Java classes 1086
Java classes delivered with PeopleTools 1086
mail *See Also* mail classes
OptBase *See Also* OptBase class methods
OptEngine *See Also* OptEngine class
OptInterface
See Also OptInterface class methods
PeopleCode *See Also* PeopleCode classes
third-party Java classes 1086
Classes property 246
ClassInfo
collection *See Also* ClassInfo collection
properties 249
ClassInfo class 2794
ClassInfo collection
list of properties, returns 2794
methods 247
properties 247
ClassInfo properties 249
Class language construct 225
CleanOutputFile method
CONQRSMGR class 673
ClearAEAttributes method
IBInfo class 1341
ClearAttachments method
IBInfo class 1342
ClearAttributes method
IBInfo class 1342
ClearConnectorProperties method 1376
ClearContainerAttributes method
IBInfo class 1343
ClearDeletesChanges method 2129
ClearDropDownList method 938
Clear method
Business Interlink 324
Response class 1075
ClearQueryStringArgs method 1377
ClientPlatform property 2277
ClientTimeZone property 2198
clone method
DataSourceParameter class 833
DataSourceParameterValue class 842
DataSourceSetting class 846
Clone method
Array class 265
Message class 1258
Close method 2601
AESection class 6
CONQRSMGR class 673
File class 984
PortalRegistry class 1616
Query class 1925
SQL class 2248
Task class 2453
Tree class 2358
Tree Structure class 2403
COBOLStmtTimings property 2204
ColImageName property 2347
Collapsed_Msg property 527
CollapsedImage property 527
CollapseMainIconSpace property 527
CollapseNode method 36
Collection class
about 753
list of methods 2690
list of properties 2691
Collection class methods

AppendItem 753
CreateItem 754
DeleteItem 755
GetItem 756
GetParent 756
GetPath 757
InsertItem 757
Collection class properties
CollectionElementType 758
Count 759
DefinedMaxOccurs 759
DefinedMinOccurs 759
ElementType 759
IsChanged 759
IsInitialized 760
IsRequired 760
Name 760
SequenceNumber 760
CollectionElementType property
Collection class 758
collections *See* PeopleCode collections
colors
selecting for charts 390, 434
ColumnLayout property
TabDefinition class 1740
UserTab class 1791
ColumnNumber property 1987
Column property
AssignedPagelet class 1749
AvailablePagelet class 1759
SelectedPagelet class 1797
comments
Incoming Business Interlink class
AddComment method 358
Task class Comments property 2458
understanding application class 221
WorklistEntry class commentshort property
1477
XmlNode class InsertComment method 2577
commentshort property 1477
Comments property
AnalyticTypeDefn class 182
CubeCollectionDefn class 122
CubeDefn class 106
DimensionDefn class 94
FeedEntry class 925
OrganizerDefn class 129
Task class 2458
UserFunctionDefn class 124
CommitWork function 1490
CompareFields method 2080
compareOp property 2626
comparison class
creating 133
GenerateRule method 134
methods 134
Operand1 property 135
Operand2 property 135
properties 135
Type property 136
CompIntfPropInfo
class 2798
collection 2798
CompIntfPropInfoCollection collection
methods
See Also CompIntfPropInfoCollection
collection methods
properties 648
CompIntfPropInfoCollection collection methods
First 647
Item 648
Next 648
CompIntfPropInfoCollection objects
Format property 649
IsCollection property 650
IsReadOnly property 650
Key property 650
LabelLong property 651
LabelShort property 651
Length property 651
Name property 652
Prompt property 652
properties 649
PropertyInfoCollection property 652
Required property 652
Type property 653
understanding 644
Xlat property 653
YesNo property 654
CompIntfPropInfoCollection property 644
component buffers 1248
ComponentId property 1464
Component Interface
accessing structure 644
classes *See Also* Component Interface classes
collection
See Also Component Interface collection
creating new instances 654
examples 654
filtering 2270
getting instances 657
GUI/online processing 611
implementing definitions 607
inserting/deleting row data 666
inserting effective-dated data 661
inserting effective-dated data via Visual
Basic 664
life cycle 600
mobile PeopleCode 1389
objects
See Also Component Interface objects
processing Search dialog boxes 611
retrieving instance lists 659
synchronizing 1386
traversing via data collections 608
understanding 599
using %Menu conditions 612
using component variables 606
using mobile objects 1389
using related languages 606
using the SetCursorPos method 945
Component Interface class
list of methods, properties, returns 2796
methods
See Also Component Interface class
methods
properties
See Also Component Interface class
properties
Component Interface classes
Component Interface
See Also Component Interface class
Data Item *See Also* Data Item
list of methods, properties, returns 2796
methods
See Also Component Interface class

methods
properties
See Also Component Interface class properties
reusing existing code 611
Session *See Also* Session class
setting keys 602
understanding 599
Component Interface classes methods
setting timeouts 607
understanding 599
using effective-dated data 609
using GetEffectiveItem 610
using InsertItem 610
using standard/user-defined 603
Component Interface classes properties
accessing standard properties 604
accessing user–defined properties 605
executing a Component Interface 604
retrieving Component Interface information 604
understanding 599
Component Interface class methods
Cancel 615
CopyRowset 616
CopyRowsetDelta 618
CopySetupRowset 620
CopySetupRowsetDelta 623
Create 626
Find 627
Get 627
GetPropertyByName 628
Save 630
SetPropertyByName 631
Component Interface class properties
ComponentName 632
CreateKeyInfoCollection 632
EditHistoryItems 632
FindKeyInfoCollection 633
GetDummyRows 633
GetHistoryItems 634
GetKeyInfoCollection 634
InteractiveMode 635
PropertyInfoCollection 635
StopOnFirstError 635
Component Interface collection
list of methods, properties, returns 2796
methods
See Also Component Interface collection methods
properties
See Also Component Interface collection properties
Component Interface collection properties
CompIntfPropInfoCollection 644
Count property 615
Component Interface collections
CompIntfPropInfoCollection
See Also CompIntfPropInfoCollection collection
Data *See Also* Data collection
Component Interface objects
declaring/scope 606
understanding 599
ComponentName property 632
components
buffers 1248
Component Interface
See Also Component Interface
component processor 383
component variables 606
Compound class
about 748
list of methods 2690
list of properties 2690
Compound class methods
GetParent 748
GetPath 749
GetPropertyByIndex 749
GetPropertyByName 750
GetUniqueKey 751
Compound class properties
ElementType 751
IsChanged 752
IsInitialized 752
IsRequired 752
Name 752
PropertyCount 752
SequenceNumber 753
CompressionOverride property 1362
ConflictAlgorithm property 2277
ConflictStatus property 2277
ConfProp property
PDFMerger class 2633
ConnDisRollbackCommit property 2204
connected query
classes *See Also* connected query classes
connected query classes
about 671
CONQRS_CONST class, about 699
CONQRS_CONST class properties 699
CONQRSMGR class, about 672
CONQRSMGR class constructor 672
CONQRSMGR class import statements 672
CONQRSMGR class methods 673
CONQRSMGR class properties 684
CONQRSPROPERTY class, about 698
CONQRSPROPERTY class properties 698
examples 713
importing 672
list of constructors, methods, properties, returns 2681
QUERYITEMPROMPT class, about 705
QUERYITEMPROMPT class properties 705
SCHED_INFO class, about 706
SCHED_INFO class properties 706
SEC_PROFILE class, about 708
SEC_PROFILE class constructor 709
SEC_PROFILE class import statements 709
SEC_PROFILE class properties 709
UTILITY class, about 710
UTILITY class constructor 710
UTILITY class import statements 710
UTILITY class methods 710
Connect method 2184
ConnectorClassName property 1382
ConnectorName property 1383
ConnectorOverride property 1363
ConnectorRequest method 1314
ConnectorRequestUrl method 1315
CONQRS_CONST class
about 699
list of properties 2684
CONQRS_CONST class properties
InitExisting 699
InitNew 700

MsgSet 700
RunCntlId_Auto 700
RunMode_Prev 701
RunMode_Prev_DefRowNumber 701
RunMode_Sample 702
RunMode_Sched_File 702
RunMode_Sched_Web 702
RunMode_Window 703
RunMode_XMLFormattedString 703
SchedRequest_CQR 704
SchedRequest_XMLP 704
Stat_Active 704
Stat_InActive 704
Stat_InProgress 704
CONQRSMGR class
about 672
import statements 672
list of methods 2681
list of properties 2682
CONQRSMGR class methods
CleanOutputFile 673
Close 673
CONQRSMGR 672
constructor 672
CopyDefn 674
DeleteDefn 674
ExistsByName 675
GetSampleXMLString 675
Open 676
ResetQueriesPrompt 677
Run 677
RunToWindow 678
RunToXMLFormattedString 679
Save 680
SaveRunControlParms 681
SetRunControlData 681
Validate 682
ValidateIfFieldsMapped 683
ValidateRunControlParms 683
CONQRSMGR class properties
Const 684
Description 684
Details 684
ErrString 685
ExecutionLog 685
IgnoreRelFldOutput 685
IsChanged 686
IsDebugMode 686
IsInit 686
IsPublic 687
LastUpdatedBy 687
LastUpdateDTTM 687
MaxRowsPerQuery 687
Name 687
ObjectRowset 688
OprID 688
OutProcessFileName 688
PropertyArray 689
QueriesPromptsArray 691
QueryArray 692
RegisteredBy 692
RegisteredDTTM 692
RunCntlId 692
RunControlParArray 693
RunMode 693
SchedInfo 695
SchedRequestType 696
SecProfile 696
ShowFormattedXML 696
Status 697
XMLDataFileName 697
XMLDataFullName 697
CONQRSMGR method
CONQRSMGR class 672
CONQRSPROPERTY class
about 698
list of properties 2684
CONQRSPROPERTY class properties
PROPDEFAULT 698
PROPNAME 698
PROPVALUE 698
PROPVALUEARRAY 699
PROPVALUEARRAYDESC 699
constant class
creating 136
methods 137
properties 138
Type property 138
using 132
Value property 138
constants
declaring 216
Expr2Constant1 property 2005
Expr2Constant2 property 2005
using 212
Const property
CONQRSMGR class 684
constructors
analytic type classes 172
BPEL classes
See Also BPEL class constructors
mail classes 1115
notifications classes
See Also Notification class constructors
PageNumber class 2627
PDFMerger class 2631
PrcsApi class 1882
Properties class 2635
Report class 2614
ReportDefn class 2600
ReportManager class 2618
SearchAttribute class 2625
understanding 204
universal queue classes
See Also universal queue classes constructors
using %This 211
Watermark class 2639
container messages
alias names 1295
copying parts 1259
determining parts 1303
determining structured parts 1303
generating XML part string 1271
getting part alias names 1274
getting part name 1274
getting part rowset 1275
getting part version 1276
getting XML 1276
part count 1306
populating parts 1282
content-based routing 1254
contentId property 2615
ContentLanguage property 1190
ContentProvider class
adding a ContentProvider 1810

ContentProvider property
Content Reference class 1688
ContentReference links 1726
Pagelet class 1779
ContentReference class 2818
Content Reference class
adding content references 1814
methods
See Also Content Reference class methods
properties
See Also Content Reference class properties
understanding 1683
Content Reference class methods
CreateLink 1683
Save 1684
Content Reference class properties
AbsoluteContentURL 1685
AbsolutePortalURL 1685
AssignedPagelets 1686
Attributes 1686
Author 1686
AuthorAccess 1687
Authorized 1687
CascadedPermissions 1687
CascadedRolePermissions 1688
ContentProvider 1688
CreationDate 1689
Data 1689
Description 1689
HtmlText 1690
IsMobile 1690
IsVisible 1690
Label 1691
Links 1691
Name 1691
OwnerId 1691
ParentName 1692
Path 1692
Permissions 1692
Product 1693
PublicAccess 1693
QualifiedURL 1693
RelativeURL 1693
RolePermissions 1694
SequenceNumber 1694
StorageType 1695
Template 1696
TemplateObject 1696
TemplateType 1697
URL 1697
URLType 1698
UsageType 1699
ValidFrom 1699
ValidTo 1700
ContentReference collection 2820
Content Reference collection
methods
See Also Content Reference collection methods
properties 1704
understanding 1700
Content Reference collection methods
DeleteItem 1700
First 1701
InsertItem 1702
ItemByName 1703
Next 1703
ContentReference Link class 2823
ContentReference links
methods 1722
properties 1723
understanding 1721
content reference links
class *See Also* ContentReference class
objects 1721
using FindCRefLinkByName method 1627
ContentReference links properties
AbsoluteContentURL 1723
AbsolutePortalURL 1724
Attributes 1724
Author 1724
AuthorAccess 1724
Authorized 1725
CascadedPermissions 1725
CascadedRolePermissions 1726
ContentProvider 1726
CreationDate 1726
Data 1726
Description 1727
IsMobile 1727
IsVisible 1727
Label 1728
Name 1728
OwnerId 1728
ParentName 1729
Path 1729
Permissions 1729
Product 1730
PublicAccess 1730
RelativeURL 1730
RolePermissions 1731
SequenceNumber 1731
Template 1731
TemplateObject 1732
TemplateType 1732
URL 1733
URLType 1733
UsageType 1733
ValidFrom 1734
ValidTo 1734
content reference objects
using FindCRefByName method 1623
using FindCRefByURL method 1624, 1626
content reference properties
TemplateType 1606
understanding 1604
URL 1607
URLType 1606
UsageType 1604
content references
adding 1814
class *See Also* Content Reference class
collection
See Also Content Reference collection
deleting content 1609
links *See Also* content reference links
naming 1608
Permissions property 1593
properties
See Also content reference properties
types 1604
understanding 1588
understanding nodes 1589
ContentRefs property 1668
Content RowsetCache property 2177

ContentType property
    MCFBodyPart class 1124
    MCFOutboundEmail class 1190
    Notification class 1454
ContentTypes property 1143
ContentURI property
    Node class 1646
    Request class 1068
ContentUrl property
    FeedDoc class 911
    FeedEntry class 925
Context property 769
Contributors property
    FeedEntry class 925
ConversationID property 1363
conversion, PeopleCode/Java 1094
convertFeedLinksToHoverMenu method
    FeedFactory class 799
convertFeedLinksToOPML method
    FeedFactory class 800
Cookie class
    list of properties, returns 2724
    properties *See Also* Cookie class properties
    understanding 1061, 1083
Cookie class properties
    Domain property 1083
    MaxAge property 1083
    Name property 1084
    Path property 1084
    Secure property 1084
    Value property 1084
cookie objects
    declaring 1060
    understanding 1061
cookies
    Cookie class *See Also* Cookie class
    cookie objects *See Also* cookie objects
    IBConnectorInfo collection Cookies property 1383
    Message class Cookies property 1296
Cookies property
    IBConnectorInfo collection 1383
    Message class 1296
CopyChangedFieldsTo method 2081
CopyCubeCollection method 64
CopyCube method 63
CopyDefn method
    CONQRSMGR class 674
CopyDimension method 65
CopyExplicitDimensionSet method 65
CopyFieldsTo method 2082
copying
    analytic instances 18
    analytic type definitions 175
    cube collections 64
Copy method
    AnalyticInstance class 18
    Tree class 2359
    Tree Structure class 2403
CopyNode method 2564
CopyObject method 1617
CopyPartRowset Method 1259
CopyPrivateQuery method 1925
Copyright property
    FeedDoc class 911
    FeedEntry class 926
CopyRowsetDelta method
    Component Interface class 618
    Message class 1261
CopyRowsetDeltaOriginal method 1263
CopyRowset method
    Component Interface class 616
    Message class 1260
    XmlDoc class 2536
CopySetupRowsetDelta method 623
CopySetupRowset method 620
CopyTo method
    AnalyticModelDefn class 66
    AnalyticTypeDefn class 175
    Row class 2113
    Rowset class 2131
CopyToPSFTMessage method 2537
CopyToRowset method 2539
CopyURLLink property 1582
CopyUserFunction method 67
Count property
    AssignedPagelet collection 1754
    Attribute collection 1711
    AvailableCategory collection 1758
    AvailablePagelet collection 1763
    Bindings collection 242
    Branch collection 2302
    CI collection 1425
    ClassInfo collection 247
    Collection class 759
    CompIntfPropInfoCollection collection 649
    Component Interface collection 615
    Content Reference collection 1704
    Data collection 643, 1431
    DynamicCategory collection 1767
    Favorite collection 1808
    Folder collection 1682
    Level collection 2322
    Link collection 1736
    MethodInfo collection 250
    Namespaces collection 244
    Node collection 1651
    NodeTemplate collection 1663
    PageletCategory collection 1777
    Pagelet collection 1788
    PermissionValue collection 1716
    Portal collection 1658
    PortalRegistry collection 1645
    PropertyInfo collection 253
    PropertyInfoCollection class 1418
    PSMessages collection 1902, 2194
    Query collection 1922
    QueryCriteria collection 1997
    QueryDBRecord collection 2045
    QueryDBRecordField collection 2052
    QueryExpression collection 2018
    QueryField collection 1983
    QueryList class 2026
    Query Metadata collection 2034
    QueryPrompt collection 2061
    QueryRecord collection 1976
    QueryRecordHierarchy collection 2030
    QuerySelect collection 1960
    RemoteNode collection 1654
    RolePermissionValue collection 1721
    SearchField collection 2490
    SearchIndexes collection 2493
    Search Language collection 2508
    SearchRecordField collection 2503
    SearchResult collection 2487
    Search Schedule collection 2512

Search Start Options collection 2523
SelectedPagelet collection 1802
TabDefinition collection 1749
UserTab collection 1796
CreateContentRefLink method 1619
CreateCookie method 1075
CreatedDateTime property 2615
CreateDocumentElement method 2541
CreateDocumentType method 2542
CreateDtTm property 1947
CreateDTTM property
Feed class 790
CreateElement method 362
createFeed method
FeedFactory class 801
CreateItem method
Collection class 754
CreateKeyInfoCollection property 632
CreateLink method 1683
Create method
AnalyticModelDefn class 68
AnalyticTypeDefn class 174
Component Interface class 626
Level class 2323
Mobile class 1399
PortalRegistry class 1618
Query class 1926
Tree class 2360
Tree Structure class 2404
WorklistEntry class 1468
CreateNextSegment method 1266
CreateNode property
Feed class 790
CreateOprId property 1947
CreateOprID property
Feed class 790
CreateOptEngine function
loading analytic instances 1491
programming for database updates 1495
running optimization transactions 1492
understanding 1508
using optimization PeopleCode in Application Engine programs 1490
using optimization PeopleCode on application servers 1489
CreateOptInterface function 1510, 1543
CreatePortal property
Feed class 790
CreateQuarantineFolder method 1132
CreateRemote method 1620
CreateRowset method 984
CreateUserId property 1947
CreateXmlDoc function 2531
creating charts 378
CreationDate property
Content Reference class 1689
ContentReference links 1726
Folder class 1668
PageletCategory class 1770
Pagelet class 1780
TabDefinition class 1740
CRefName property 1802
criteria
AddHavingCriteria method 1963
adding 2068
Criteria property 1970
DeleteHavingCriteria 1968
having criteria for queries 1903
HavingCriteria property 1971
queries 1894
Criteria property 1970
CrumbDescrSelectStyle property 542
CrumbDescrStyle property 528
CrumbMaxDisplayLength property 528
CrumbSeparatorImage property 528
Crypt
class *See Also* Crypt class
objects 721
Crypt class
functions, methods, properties, returns 2686
methods *See Also* Crypt class methods
properties 727
understanding 721
Crypt class methods
FirstStep 722
GoToStep 723
LoadLibrary 723
NextStep 724
Open 725
SetParameter 725
UpdateData 726
Crypt class properties 727
Crypt objects 721
cryptography 721
cube
accessing 74
names 77
causes 100
circular dependencies 101
dimension aggregates 101
effects 102
dimension names 103
rules 103
adding to analytic model 59
attaching dimensions 98
attaching to cube collections 108
calculating 28
calculating aggregates 106
class methods 98
class properties 106
commenting 106
copying 63
count for collections 122
counting dimensions 107
deleting 69
detaching dimensions 99
detaching from cube collections 109
naming 107
setting
rules 105
specifying format type 107
using dimensions 105
value dimension names 108
virtual 107
cube class
AddIndex method 139
creating 139
GenerateRule method 140
GetIndexes method 140
methods 139
Name property 141
property 141
cube collection
accessing
dimension sort settings 42
current layout 43

names 111
dimension names 111
adding to analytic model 59
aggregate record name 122
analytic grids 164
attaching cubes 108
class methods 108
class properties 121
collapsing nodes 36
commenting 122
copying 64
counting cubes 122
counting dimensions 122
deleting 70
detaching cubes 109
DrillIntoNode 37
DrillOutOfNode 38
expanding nodes 39
filtering 114, 118
getting a slice 44
getting field mapping 113
getting names 76
getting row count 44
hierarchy display 49
messages property 52
naming 122
persist aggregate 114
populating from rowset 45
populating rowsets 40
record name 123
removing filters 50
removing sorting 51
renaming 85
returning user function 41
setting aggregate mapping 115
setting dimension order 46
setting field mapping 117
setting layout 166
sorting dimensions 112, 116
specifying slice records 48
specifying sort order 47
using cubes 120
using dimensions 121
CubeCollection class 36
CubeCollection class property 51
CubeCollectionDefn class
methods 108
properties 121
CubeCollectionDefn class methods 108
CubeCollectionDefn class properties 121
CubeCount property 122
cube definition
aggregate mapping 110
CubeDefn class
methods 98
properties 106
CurrencyFormat property 2199
CurrencySymbol property 2199
CurrentItem method 636
CurrentItemNumber property 643
CurrentRecord property 1009
CurrentSegment property 1297
cursors
ResetCursor method 356
ReuseCursor property 2256
reusing 2244
SetCursorPos method 945
CustomizePageLink property 1583
custom runtime prameters, XML Publisher
setting 2610
Cut method
Leaf class 2303
Node class 2327

D

data
collection *See Also* Data collection
Content Reference class Data property 1689
ContentReference links Data property 1726
Data Item class 643
types *See Also* data types
DataAreaCollapsed property
Field class 947
Rowset class 2162
database cache *See* RowsetCache class
DatabaseName property
Report class properties 2615
databases
forcing a commit on updates
1489, 1490, 1495
programming for updates 1495
queries 1892
Data collection
list of methods, properties, returns
2799, 2877
methods
See Also Data collection methods, Data
collection methods
objects 1425
properties 642, 1430
traversing a Component Interface 608
understanding 1394
Data collection methods
CurrentItem 636
DeleteItem 636, 1428
GetEffectiveItem 638
GetEffectiveItemNum 639
GetListViewAttrs 1426
InsertItem 640, 1429
Item 641, 1427
ItemByKeys 641
Data collection objects 1425
DataEndDateTime property 507
data hints 439
Data Item class 643
Data property
Content Reference class 1689
ContentReference links 1726
DataSource class
about 815
import statements 815
list of methods 2697
list of properties 2697
DataSource class methods
addParameter 816
constructor 815
DataSource 815
equals 817
getAttributeById 818
getContentUrl 819
getDataSecurity 819
getParameterById 820
getParameterDetail 821

getSettingById 821
getSettingDetail 822
isCurrentUserAdmin 823
isCurrentUserAuthorized 823
onDelete 824
onSave 825
resetParameters 825
DataSource class properties
AdminPersonalizationPage 826
AllowCustomParameters 826
AllowRealTimeFeedSecurity 827
DataSourceType 827
DefaultFeedAttributes 827
DefaultIBOperationName 827
Description 827
HasParameters 828
HasSettings 828
IBOperations 828
ID 828
LongDescription 828
ObjectType 829
Parameters 829
ParametersCompleted 829
Parent 829
PortalSpecificPersonalization 830
Settings 830
SettingsCompleted 830
UserPersonalizationPage 830
Utility 831
DataSource method
DataSource class 815
DataSourceParameter class
about 831
import statements 831
list of methods 2699
list of properties 2699
DataSourceParameter class methods
addUserValue 832
clone 833
constructor 831
DataSourceParameter 831
equals 833
resetUserValues 834
setRangeFromFieldTranslates 834
validateValue 835
DataSourceParameter class properties
AllowChangesToRequired 836
DefaultValue 836
DefaultValueForDisplay 836
Description 836
EditType 836
EvaluatedValue 837
FieldType 837
ID 838
Name 838
ObjectType 838
Parent 838
PromptTable 839
Range 839
Required 839
SkipValidityChecks 839
UsageType 840
UserValues 840
Utility 840
Value 841
ValueForDisplay 841
DataSourceParameter method
DataSourceParameter class 831
DataSourceParameterValue class
about 841
list of methods 2700
list of properties 2700
DataSourceParameterValue class methods
clone 842
constructor 842
DataSourceParameterValue 842
equals 843
DataSourceParameterValue class properties
Description 844
ID 844
Name 844
ObjectType 844
OrderNumber 844
Parent 845
Value 845
DataSourceParameterValue method
DataSourceParameterValue class 842
DataSource property
Feed class 790
DataSourceSetting class
about 845
import statements 846
list of methods 2701
list of properties 2701
DataSourceSetting class methods
clone 846
constructor 846
DataSourceSetting 846
equals 847
setRangeFromFieldTranslates 847
DataSourceSetting class properties
DefaultValue 848
DisplayOnly 848
EditType 848
Enabled 849
FieldType 849
ID 850
LongName 850
Name 850
ObjectType 850
Parent 850
PromptTable 851
Range 851
RefreshOnChange 851
RelatedFieldValue 851
Required 852
ShowRelatedField 852
Utility 852
Value 853
Visible 853
DataSourceSetting method
DataSourceSetting class 846
DataSources property
FeedFactory class 814
DataSourceType property
DataSource class 827
DataStartDateTime property 507
DataStartRow property 454, 507
DataTypeID property
Feed class 791
data types
declaring array objects 264
declaring BPEL 288
declaring Business Interlink objects 311
declaring chart objects 397, 406
declaring Component Interface objects 606

declaring Cookie objects 1060
declaring Exception objects 765
declaring field objects 936
declaring file objects 976
declaring Grid/GridColumn objects 1037
declaring iScript objects 1060
declaring Java objects 1099
declaring MCFIMInfo objects 1234
declaring message objects 1247
declaring mobile objects 1397
declaring notifications 1446
declaring OrgChart objects 425
declaring page objects 1582
declaring PortalRegistry objects 1610
declaring PostReport objects 1826
declaring ProcessRequest objects 1836
declaring query objects 1903
declaring RatingBoxChart objects 430
declaring record objects 2079
declaring Request objects 1060
declaring Response objects 1060
declaring row objects 2112
declaring RowsetCache objects 2173
declaring rowset objects 2128
declaring session objects 2183
declaring SOAPDoc objects 2218
declaring SQL objects 2247
declaring SyncServer objects 2271
declaring Tree objects 2297
declaring XmlDoc objects 2535
declaring XmlNode objects 2535
mail classes 1102
mapping PeopleCode and Java 1094
passing parameters 200
universal queue classes 2422
XML Publisher objects 2598
DataWidth property 454, 507
DateFormat property 2199
dates
effective dates *See Also* effective dates
using with ReadRowset method 995
DateSeparator property 2200
dateStringToUserPref method
Utility class 854
datetimeToString method
FeedDoc class 905
Utility class 855
dateToString method
Utility class 855
DB2 UDB for OS/390 and z/OS
considerations for extra spaces 2240
sorting records 2136
DBPatternMatch method 2084
DBRecordName property 2163
DBSpecificCalls property 2204
DeactivateModel method 1567
debugging
mobile PeopleCode 1394, 1395
setting trace options 2182
using the Java Debugging Environment (JDB) 1098
DecimalPosition Field class 948
Decimal property
ColumnNumber class 1987
QueryDBRecordField class 2053
QueryExpression class 2020
DecimalSymbol property 2200
DecodeText method 1161
DecodeWord method 1163
decodeXML method
Utility class 856
DecryptPETKey method 939
DefaultCharSet property 1191
DefaultChartNavigation property
Folder class 1669
DefaultColumn property 1780
DefaultFeedAttributes property
DataSource class 827
DefaultIBOperationName property
DataSource class 827
DefaultImage property 529
DefaultMessageVersion property 1297
DefaultPortalName property 1646
DefaultStyleSheetName property 1082
DefaultTemplate property
NodeTemplate class 1658
PortalRegistry class 1639
DefaultText property 769
DefaultValueForDisplay property
DataSourceParameter class 836
DefaultValue property 1419
DataSourceParameter class 836
DataSourceSetting class 848
DefinedMaxOccurs property
Collection class 759
DefinedMinOccurs property
Collection class 759
DeleteAEAttribute method
IBInfo class 1344
DeleteAll method 2191
DeleteAttachment method
IBInfo class 1344
DeleteAttribute method
IBInfo class 1345
DeleteByName method 2328
DeleteByRange method 2304
deleteCategory method
FeedDoc class 906
FeedEntry class 922
DeleteConnectorProperties method 1377
DeleteContainerAttribute method
IBInfo class 1346
deleteContributor method
FeedEntry class 922
DeleteCriteria method 1966
DeleteCubeCollection method 70
DeleteCube method 69
DeleteDefn method
CONQRSMGR class 674
DeleteDimension method 71
DeleteEmail method 1158
DeleteEnabled property
Row class 2119
Rowset class 2163
deleteEnclosure method
FeedEntry class 923
deleteEntry method
FeedDoc class 907
DeleteExplicitDimensionSet method 71
DeleteExpression method 1967
deleteFeed method
FeedFactory class 802
DeleteField method 1968
DeleteHavingCriteria method 1968
DeleteHomepage method 1622
DeleteItem method

AssignedPagelet collection 1751
Attribute collection 1707
Collection class 755
Content Reference collection 1700
Data collection 636, 1428
DynamicCategory collection 1763
Favorite collection 1804
Folder collection 1679
NodeTemplate collection 1659
PageletCategory collection 1773
Pagelet collection 1785
PermissionValue collection 1713
PSMessages collection 2192
RolePermissionValue collection 1717
Search Language collection 2506
SearchRecordField collection 2501
Search Schedule collection 2510
Search Start Options collection 2521
SelectedPagelet collection 1798
TabDefinition collection 1745
UserTab collection 1793
delete method
Feed class 777
FeedEntry class 921
Delete method
Agent class 2434
AnalyticInstance class 19
AnalyticModelDefn class 68
analytic type definition 176
ContentReference links 1722
File class 986
Leaf class 2303
Node class 2327
PortalRegistry class 1621
Query class 1928
Record class 2085
RowsetCache class 2173
Tree class 2362
Tree Structure class 2405
DeleteModel method 176
DeleteOptProbInst function 1494, 1495, 1511
DeleteOrganizer method 72
DeleteOrphanedSegments method 1316
DeletePrompt method 1929
DeleteQuerySelect method 1957
DeleteQueryStringArg method 1378
DeleteRecord method 177, 1969
DeleteRow method 2133
DeleteSegment method 1267
DeleteUserFunction method 73
DeliveryMode property 1363
delivery status notifications *See* email
deprecated
classes 2871
functions 2871
methods 2871
properties 2871
deprecated functions 2881
Description property
AnalyticModelDefn class 91
AnalyticTypeDefn class 183
AnalyticTypeRecordDefn class 186
ColumnNumber class 1987
CONQRSMGR class 684
Content Reference class 1689
ContentReference links 1727
DataSource class 827
DataSourceParameter class 836
DataSourceParameterValue class 844
Feed class 791
FeedDoc class 911
FeedEntry class 926
Folder class 1669
Leaf class 2315
Level class 2324
MCFBodyPart class 1124
MCFOutboundEmail class 1191
Node class 1647, 2347
NotificationAddress class 1460
PageletCategory class 1770
Pagelet class 1780
PortalRegistry class 1640
Query class 1948
QueryDBRecord class 2047
QueryDBRecordField class 2053
QueryRecord class 1978
QueryRecordHierarchy class 2030
Report class 2616
ReportDefn class 2611
RowsetCache class 2177
TabDefinition class 1740
Tree class 2387
Tree Structure class 2407
DestinationNode property 1364
DestinationPrinter property 2611
DestMsgName property 2288
DestMsgVersion property 2288
DestNode property 2288
DetachCube method 109
DetachDimension method
cube class 99
ExplicitDimensionSet class 96
DetachPart method 126
DetachTree methods 29
DetailComponent property 2407
DetailedStatus property 1542
DetailField property 2408
DetailMenuBar property 2409
DetailMenuItem property 2409
DetailMenu property 2408
DetailMsgs property 1541
DetailMultiNavigate property 2409
DetailPage property 2410
DetailRecord property 2410
Details property
CONQRSMGR class 684
DetailViewLabel property 1432
DigitGroupingSymbol property 2200
dimension
accessing
names 78
explicit set names 80
aggregate sequence 94
names 103
adding to analytic model 60
analytic model 77
attaching to cubes 98
class properties 93
commenting 94
copying 65
deleting 71
detaching from cubes 99
filter, setting 46
names for cube collection 111
naming 94
order, setting 46

renaming 85
setting aggregates 104
sorting
CubeCollection class 47
CubeCollectionDefn class 112, 116
sort setting 42
using 105, 121
using total values 95
DimensionCount property
CubeCollectionDefn class 122
CubeDefn class 107
DimensionDefn class
properties 93
Dimension property
Array class 285
MemberReference class 150
dimensions
attaching 95
detaching 96
Direction property 529
DIRLOCATION property
SCHED_INFO class 707
Disabled property 1948
Disconnect method 2186
Discovery (API) 234
DisplayFormat property 948
DisplayLevelNumber property
Leaf class 2315
Node class 2347
DisplayOnly property
DataSourceSetting class 848
Field class 950
Page class 1583
PeerDefaultAttributes class 1440
PropertyAttrs class 1436
DisplayOutput method 2602
DisplayZeroChanged property 950
DisplayZero property 950
Disposition property
MCFBodyPart class 1125
MCFOutboundEmail class 1191
Distinct property 1970
DistributeDataByRules function 2267
distribution lists, determining 1825
do_replicate_flag property 1477
document
classes *See Also* document classes
documentation *See* comments
Documentation property
ClassInfo class 249
MethodInfo class 252
PropertyInfo class 255
Document class
about 730
list of methods 2688
list of properties 2688
document classes
about 729
built-in functions 730
Collection class, about 753
Collection class methods 753
Collection class properties 758
Compound class, about 748
Compound class methods 748
Compound class properties 751
data types 729
Document class, about 730
Document class methods 730
Document class properties 738
DocumentKey class, about 739
DocumentKey class methods 739
DocumentKey class properties 740
list of functions, methods, properties, returns 2687
Primitive class, about 741
Primitive class methods 741
Primitive class properties 743
scope 729
Document class methods
GenXmlString 730
GetDocumentKey 731
GetElement 732
GetRowset 733
GetSchema 734
ParseXmlFromFile 735
ParseXmlFromURL 735
ParseXmlString 736
UpdateFromRowset 737
ValidateData 737
Document class properties
DocumentElement 738
DocumentElement property 2552
Document class 738
DocumentKey class
about 739
list of methods 2688
list of properties 2688
DocumentKey class methods
SetDocumentKey 739
DocumentKey class properties
DocumentName 740
PackageName 740
Version 740
DocumentName property
DocumentKey class 740
DomainLimit property 2515
Domain property 1083
DoNotPubToNodeName property 1298
DoSave function 1444
DoSaveNow function 1490
downcasting 219
DraggedNodeStyle property 547
DrillIntoNode method 37
DrillOutOfNode method 38
DropDownBoxStyle property 529
DSPARAMETER_INCREMENTAL property
Utility class 875
DSPARAMETER_MAXROW property
Utility class 876
DSPARAMETER_SF_MAXMINUTES property
Utility class 876
DSPARAMETER_SF_PAGING property
Utility class 876
dttmCreated property 1454
DttmReceived property 1153
DttmSaved property 1153
DttmSent property 1153
DumpMsgToLog method 1567
DumpToFile method 1147
DuplicateLeaves property 2388
DynamicCategories property 1740
DynamicCategory collection
list of methods, properties, returns 2829
methods
See Also DynamicCategory collection methods

properties 1767
understanding 1763
DynamicCategory collection methods
DeleteItem 1763
First 1764
InsertItem 1765
ItemByName 1765
Next 1766
dynamic output
Business Interlink objects 305
GetFieldCount method 338
GetFieldType method 339
GetFieldValue method 340
MoveFirst method 352
MoveNext method 353
Dynamic property 2315

E

EditError property 951
EditHistoryItems property 632
EditPageContentProvider property 1780
EditPageQueryString property 1781
EDITTYPE_NOTABLEEDIT property
Utility class 877
EDITTYPE_PROMPTTABLE property
Utility class 877
EDITTYPE_TRANSLATETABLE property
Utility class 877
EDITTYPE_YESNO property
Utility class 877
EditType property 2061
DataSourceParameter class 836
DataSourceSetting class 848
EffDt property
Rowset class 2164
Tree class 2388
effective dates
editing records 1269
inserting data in component interfaces 661
inserting data in component interfaces via Visual Basic 664
inserting rows 2149
using the Record class SetEditTable method 2087
using with a Component Interface 609
effects 102
EffSeq property 2165
ElementType property
Collection class 759
Compound class 751
Primitive class 743
email
delivery status notifications (DSN) 1105
double-downloading 1109
errors/status 1108
instant messaging *See Also* instant messaging
mail classes *See Also* mail classes
MIME 1108
outbound 1104
priority 1105
retrieving 1107
return receipts 1105
rowset structure 1108
setting attributes 1844
setting languages *See Also* languages
EmailAttachLog property 1865
EmailId property 1460
EmailReplyTo property 1455
EmailSubject property 1866
EmailText property 1866
EmailWebReport property 1867
EnableColumns method
Grid class 1038
Enabled property
DataSourceSetting class 849
Field class 952
GridColumn class 1046
Enclosures property
FeedEntry class 926
EncodeText method 1164
EncodeWord method 1165
encodeXML method
Utility class 857
encryption profiles 721
EncryptPETKey method 940
engine classes
about 2596, 2627
engines
application *See Also* Application Engine
EnqueuedTaskList property 2450
Enqueue method 2454
EnqueueTime property 2458
entities
adding, inserting, getting, finding 2533
InsertEntityReference method 2581
Entries property
FeedDoc class 912
EnumCount property
Primitive class 743
EnvelopeNode property 2231
environment variables
PS_CLASSPATH 1086
using relative paths 982
equals method
DataSource class 817
DataSourceParameter class 833
DataSourceParameterValue class 843
DataSourceSetting class 847
Feed class 777
FeedDoc class 907
FeedEntry class 924
ErrorCount property
MCFGetMail class 1143
ParseResult Collection property 2483
ErrorDescription property 1192
MCFMailUtil class 1173
ErrorDetails property 1192
MCFMailUtil class 1173
ErrorMsgParamsCount property
MCFMailUtil class 1174
MCFOutboundEmail class 1192
ErrorPending property 2188
errors
displaying error messages 2181
email 1114
Exception class *See Also* Exception class
forcing commits on pending database updates 1490
handling chart errors 397
handling for all APIs 2180
handling for iScripts 1059
handling for Java functions 1098
handling for messages 1256

handling for mobile PeopleCode 1396
handling for PortalRegistry classes 1595
handling for query classes 1901
handling for RowsetCache 2172
handling for Search classes 2470
handling for subscription PeopleCode 2180
handling for tree classes 2294
handling for XmlNode class 2534
handling for XML Publisher 2597
loading analytic instances 1491
logging API errors 2181
logging for APIs called from Application Engine 2181
logging system errors 2181
processing file layout errors 981
retrieving email 1108
specifying severity for request messages 1500
using the DumpMsgToLog method 1567
using the GetTraceLevel method 1533
using the SetTraceLevel method 1537
ErrString property
CONQRSMGR class 685
EscalatedTaskList property 2450
EscalationTime property 2458
EvaluatedValue property
DataSourceParameter class 837
evaluateSysVar method
Utility class 858
events, PeopleCode *See* PeopleCode events
events, synchronization server
See synchronization server events
Exception class
functions, methods, properties, returns 2692
methods *See Also* Exception class methods
properties
See Also Exception class properties
try-catch sections 764
understanding 220, 761
using 762
Exception class methods
GetSubstitution method 765
Output method 766
SetSubstitution method 767
ToString method 768
Exception class properties
Context property 769
DefaultText property 769
MessageNumber 769
MessageSetNumber 769
MessageSeverity property 770
StackTrace property 770
SubstitutionCount property 771
exceptions
Exception class
See Also Exception class, Exception class
handling for Java functions 1098
MCFGetMail class methods 1115
message class 1299
objects 765
throwing 762
understanding 761
using 762
using transfer functions 763
using try-catch statements 764
ExcludeProperty property 2278
ExecAppName property 1948
ExecCount property 2037
ExecLogging property 1949
ExecuteEdits method
Message class 1268
Record class 2086, 2101
execute method
Feed class 778
Execute method
Business Interlink class 325
SearchQuery class 2476
SQL class 2249
ExecutionLog property
CONQRSMGR class 685
ExistsByName method
CONQRSMGR class 675
Exists method 2363
Expanded_Msg property 530
ExpandedImage property 530
Expand method 2328
ExpandNode method 39
ExpImageName property 2348
ExpirationDate property 1828
ExpireDate property 2616
ExpireMeta property 1068
Expires property
FeedDoc class 912
FeedEntry class 927
ExplainText property 2194
ExplicitDimensionSet
AttachDimension method 95
DetachDimension method 96
GetDimensionNames method 97
Name property 97
explicit dimension sets
adding to analytic model 61
analytic model 79
copying 65
deleting 71
renaming 86
ExpNum property
ColumnNumber class 1987
QueryExpression class 2020
ExportLocation property 2278
Expr1Expression property 2003
Expr1Field property 2003
Expr1Type property 2004
Expr2Constant1 property 2005
Expr2Constant2 property 2005
Expr2Expression1 property 2005
Expr2Expression2 property 2006
Expr2Field1 property 2006
Expr2Field2 property 2006
Expr2List property 2007
Expr2Subquery property 2007
Expr2Type property 2007
ExpressionBlock class
creating 141
expression blocks
adding 142
getting 142
Expression property 133
Expressions property 1970
external functions, declaring 206
external interfaces, declaring public 193
ExternalMessageID property 1364
ExternalOperationName property 1364
ExternalUserName property 1364
ExternalUserPassword property 1365
ExtraOptions property 2495

F

FaultCode property 2231
FaultCodeS property 2232
faults
  AddFault method 2222
  FaultCode property 2231
  FaultCodeS property 2232
  FaultString property 2232
  SOAPDoc objects 2217
FaultString property 2232
Favorite class
  properties *See Also* Favorite class properties
  understanding 1802
Favorite class properties
  CRefName 1802
  Label 1803
  QualifiedURL 1803
  SequenceNumber 1803
  URL 1803
Favorite collection
  methods
    See Also Favorite collection methods
  properties 1808
  understanding 1804
Favorite collection methods
  DeleteItem 1804
  First 1805
  InsertItem 1805
  ItemByLabel 1806
  Next 1807
  Save 1807
Favorite collection property 1808
favorites
  class *See Also* Favorite class
  collection *See Also* Favorite collection
  Favorites property 1640
  objects 1802
Favorites class 2836
Favorites collection 2836
Favorites property 1640
feed
  classes *See Also* feed classes
FEEDATTRIBUTE_AUTHOR property
  Utility class 878
FEEDATTRIBUTE_CLOUD property
  Utility class 878
FEEDATTRIBUTE_COMPLETE property
  Utility class 878
FEEDATTRIBUTE_CONTRIBUTOR property
  Utility class 878
FEEDATTRIBUTE_COPYRIGHT property
  Utility class 879
FEEDATTRIBUTE_EXPIRES property
  Utility class 879
FEEDATTRIBUTE_ICONURL property
  Utility class 879
FEEDATTRIBUTE_LOGOURL property
  Utility class 879
FEEDATTRIBUTE_MANAGINGEDITOR property
  Utility class 880
FEEDATTRIBUTE_MAXAGE property
  Utility class 880
FEEDATTRIBUTE_PERSINSTRUCTION property
  Utility class 880
FEEDATTRIBUTE_RATING property
  Utility class 880
FEEDATTRIBUTE_SKIPDAYS property
  Utility class 880
FEEDATTRIBUTE_SKIPHOURS property
  Utility class 881
FEEDATTRIBUTE_TEXTINPUT property
  Utility class 881
FEEDATTRIBUTE_TTL property
  Utility class 881
FEEDATTRIBUTE_WEBMASTER property
  Utility class 881
FEEDATTRIBUTE_XSL property
  Utility class 882
FeedAttributes property
  Feed class 791
FeedAuthorizationOprID property
  Feed class 791
FeedAuthorizationOprPWD property
  Feed class 792
FeedAuthorizationType property
  Feed class 792
FEEDAUTHTYPE_ALL property
  Utility class 882
FEEDAUTHTYPE_ANONYMOUS property
  Utility class 882
FEEDAUTHTYPE_DEFAULT property
  Utility class 882
FeedCacheTime property
  Feed class 792
FEEDCACHETYPE_NONE property
  Utility class 882
FEEDCACHETYPE_PRIVATE property
  Utility class 883
FEEDCACHETYPE_PUBLIC property
  Utility class 883
FEEDCACHETYPE_ROLE property
  Utility class 883
FeedCacheType property
  Feed class 792
Feed class
  about 776
  import statements 776
  list of methods 2693
  list of properties 2694
feed classes
  about 773
  DataSource class, about 815
  DataSource class constructor 815
  DataSource class import statements 815
  DataSource class methods 816
  DataSource class properties 826
  DataSourceParameter class, about 831
  DataSourceParameter class constructor 831
  DataSourceParameter class import statements 831
  DataSourceParameter class methods 832
  DataSourceParameter class properties 835
  DataSourceParameterValue class, about 841
  DataSourceParameterValue class constructor 842
  DataSourceParameterValue class methods 842
  DataSourceParameterValue class properties 843
  DataSourceSetting class, about 845
  DataSourceSetting class constructor 846
  DataSourceSetting class import statements

846
DataSourceSetting class methods 846
DataSourceSetting class properties 848
Feed class, about 776
Feed class constructor 776
Feed class import statements 776
Feed class methods 776
Feed class properties 789
FeedDoc class, about 902
FeedDoc class constructor 903
FeedDoc class import statements 903
FeedDoc class methods 903
FeedDoc class properties 910
FeedEntry class, about 917
FeedEntry class constructor 918
FeedEntry class import statements 918
FeedEntry class methods 918
FeedEntry class properties 924
FeedFactory class, about 798
FeedFactory class constructor 799
FeedFactory class import statements 799
FeedFactory class methods 799
FeedFactory class properties 814
importing 775
list of constructors, methods, properties, returns 2693
Utility class, about 853
Utility class constructor 854
Utility class import statements 854
Utility class methods 854
Utility class properties 874
Feed class methods
constructor 776
delete 777
equals 777
execute 778
Feed 776
getAttribute 779
load 780
populatePrefData 781
publishToSites 782
resetFeedAttributes 783
resetFeedSecurities 784
save 785
saveAs 785
setAttribute 786
setDataSourceById 787
unpublishFromSites 788
Feed class properties
Authorized 789
CategoryID 789
CreateDTTM 790
CreateNode 790
CreateOprID 790
CreatePortal 790
DataSource 790
DataTypeID 791
Description 791
FeedAttributes 791
FeedAuthorizationOprID 791
FeedAuthorizationOprPWD 792
FeedAuthorizationType 792
FeedCacheTime 792
FeedCacheType 792
FeedContentUrl 793
FeedFactory 793
FeedFormat 793
FeedSecurities 793
FeedSecurityType 794
FeedUrl 794
HasAdminParams 794
HasUserParams 794
IBOperationName 795
ID 795
LastPubDTTM 795
LastUpdDTTM 795
LastUpdOprID 796
NameSpaceID 796
ObjectType 796
OperatingMode 796
OwnerID 797
PublishedInSites 797
Title 797
Utility 798
FeedContentUrl property
Feed class 793
FeedDoc class
about 902
import statements 903
list of methods 2710
list of properties 2710
FeedDoc class methods
addCategory 903
addEntry 904
constructor 903
datetimeToString 905
deleteCategory 906
deleteEntry 907
equals 907
FeedDoc 903
getEntry 908
resetEntries 909
stringToDatetime 909
FeedDoc class properties
AllowMoreEntries 910
ApplicationRelease 910
Categories 910
ContentUrl 911
Copyright 911
Description 911
Entries 912
Expires 912
FeedFormat 912
FeedUrl 913
FirstUrl 913
Generator 914
ID 914
LastUrl 914
Logo 915
MaxAge 915
MaxEntries 915
NextUrl 916
ObjectType 916
PreviousUrl 916
RootElement 917
Title 917
Updated 917
FeedDoc method
FeedDoc class 903
FeedDoc property
FeedEntry class 927
FeedEntry class
about 917
import statements 918
list of methods 2712
list of properties 2712

FeedEntry class methods
addCategory 919
addContributor 919
addEnclosure 920
constructor 918
delete 921
deleteCategory 922
deleteContributor 922
deleteEnclosure 923
equals 924
FeedEntry 918
FeedEntry class properties
Author 924
Categories 925
Comments 925
ContentUrl 925
Contributors 925
Copyright 926
Description 926
Enclosures 926
Expires 927
FeedDoc 927
FullContent 927
GUID 928
ID 928
MaxAge 928
ObjectType 929
Published 929
Title 929
Updated 929
FeedEntry method
FeedEntry class 918
FeedFactory class
about 798
import statements 799
list of methods 2696
list of properties 2697
FeedFactory class methods
constructor 799
convertFeedLinksToHoverMenu 799
convertFeedLinksToOPML 800
createFeed 801
deleteFeed 802
FeedFactory 799
genFeedUrl 802
genUniqueFeedId 803
getAllPagedFeedLinks 804
getCategory 805
getDataSource 806
getFeed 807
getFeedDoc 809
getFeedLink 810
getFeedLinks 810
getRelatedFeedsHoverMenu 811
getRelativePageLinkForFeed 812
FeedFactory class properties
DataSources 814
Feeds 814
Utility 814
FeedFactory method
FeedFactory class 799
FeedFactory property
Feed class 793
FEEDFORMAT_ATOM10 property
Utility class 883
FeedFormat property
Feed class 793
FeedDoc class 912
Feed method
Feed class 776
feeds
implementing a request handler 930
request handler, implementing 930
FeedSecurities property
Feed class 793
FeedSecurityType property
Feed class 794
FEEDSECUTYPE_PUBLIC property
Utility class 883
FEEDSECUTYPE_REALTIME property
Utility class 884
FEEDSECUTYPE_SELECTED property
Utility class 884
Feeds property
FeedFactory class 814
FEEDTYPE_DYNAMIC property
Utility class 884
FEEDTYPE_PREPUBLISHED property
Utility class 884
FeedUrl property
Feed class 794
FeedDoc class 913
FetchIntoRecord method 327
FetchIntoRowset method 329
Fetch method 2250
FetchNextRow method 332
Field class
declaring 936
functions, methods, properties, returns 2713
methods *See Also* Field class methods
properties *See Also* Field class properties
shortcut considerations 936
understanding 935
Field class methods
AddDropDownItem method 937
ClearDropDownList method 938
DecryptPETKey method 939
EncryptPETKey method 940
GetAuxFlag method 940
GetLongLabel method 941
GetRelated method 942
GetShortLabel 943
SearchClear 944
SetCursorPos 945
SetDefault 945, 946
Field class properties
DataAreaCollapsed 947
DecimalPosition 948
DisplayFormat 948
DisplayOnly 950
DisplayZero 950
DisplayZeroChanged 950
EditError 951
Enabled 952
FieldLength 952
FormatLength 952
FormattedValue 953
HoverText 954
IsAltKey 954
IsAuditFieldAdd 954
IsAuditFieldChg 955
IsAuditFieldDel 955
IsAutoUpdate 955
IsChanged 955
IsDateRangeEdit 956
IsDescKey 956

IsDuplKey 956
IsEditTable 956
IsEditXlat 957
IsFromSearchField 957
IsInBuf 957
IsKey 958
IsListItem 958
IsNotUsed 958
IsRequired 959
IsRichTextEnabled 959
IsSearchItem 959
IsSystem 959
IsThroughSearchField 959
IsUseDefaultLabel 960
IsYesNo 960
Label 960
LabelImage 961
LongTranslateValue 962
MessageNumber 962
MessageSetNumber 963
MouseOverMsgNum 963
MouseOverMsgSet 964
Name 964
OriginalValue 965
ParentRecord 965
PromptTableName 966
SearchDefault 966, 2095
SearchEdit 966
SetComponentChanged 967
ShortTranslateValue 968
ShowRequiredFieldCue 969
SmartZero 969
SqlText 969
StoredFormat 970
Style 971
Type 971
Value 972
Visible 973
FieldCount property 2104
FieldDecimal property 2062
FieldFormat property 2062
FieldLength property
Field class 952
QueryPrompt class 2063
field mapping
getting 113
setting 117
fieldname property 2104
FieldName property
QueryPrompt class 2063
SearchRecordField class 2504
field objects
declaring/scope 936
understanding 935
fields
Field class *See Also* Field class, Field class
objects 935
processing numeric fields via ExecuteEdits
1269, 2088
QueryDBRecordFields 1894
query definitions 1893
understanding 935
using fixed length files with numeric fields
1007
Fields property 2498
FIELDTYPE_CHARACTER property
Utility class 884
FIELDTYPE_DATE property
Utility class 885
FIELDTYPE_DATETIME property
Utility class 885
FIELDTYPE_LONGCHARACTER property
Utility class 885
FIELDTYPE_NUMBER property
Utility class 885
FIELDTYPE_SIGNEDNUMBER property
Utility class 885
FIELDTYPE_TIME property
Utility class 886
FieldType property 2063
DataSourceParameter class 837
DataSourceSetting class 849
File class
access interruptions 977
functions, methods, properties, returns 2718
methods *See Also* File class methods
processing errors 981
properties *See Also* File class properties
understanding 975
File class methods
Close 984
CreateRowset 984
Delete 986
GetPosition 977, 986
GetString 988
Open 989
ReadLine 993
ReadRowset 994
SetFileId 996
SetFileLayout 998
SetPosition 977, 1000
SetRecTerminator 1001
WriteLine 1002
WriteRaw 1003
WriteRecord 1004
WriteRowset 1006
WriteString 1008
File class properties
CurrentRecord 1009
IgnoreInvalidId 1010
IsError 1010
IsNewFileId 1011
IsOpen 1012
Name 1012
TerminateLines 1012
UseSpaceForNull 1013
ZeroExtend 1016
file definition tags 1004, 1007
file dependant processing
ProcessRequest class FileName property
1867
understanding 1845
using PrcsApi class 1880
File Layout definitions
accessing 980
Application Engine example 1026
examples 1016
file rowset example 1026
reading/writing hierarchical data 978
reading multiple 1030
ReadRecord example 1020
ReadRowset example 1025
understanding 975
using multiple 1029
using XML 1004, 1007
WriteRecord example 1017

WriteRowset example 1021
writing multiple 1032
Filename property
MCFBodyPart class 1125
MCFOutboundEmail class 1193
FileName property 1867
FileNames property 1455
file objects
declaring/scope 976
security 977
FilePath property
MCFBodyPart class 1125
MCFOutboundEmail class 1193
FilePathType property
MCFBodyPart class 1126
MCFOutboundEmail class 1193
files
definition tags 1004
File class *See Also* File class
file definition tags 1007
file dependant processing
See Also file dependant processing
FileName property 1867
FileNames property 1455
FileTitles property 1455
getAllFileNames method 1882
layout *See Also* File Layout definitions
objects 976
plain text 978
using file rowsets 1026
using fixed length files with numeric fields 1007
FileTitles property 1455
FileURL property 2616
FillAppend method 2138
Fill method 2135
FillRowset method 1520
filtering
SearchRecord Options class Filter property 2498
synchronization server OnGetPropertyFilter event 2270
using with synchronization 2270
Filter property 2498
filters
accessing 114
setting 118
FinalDestinationNode property 1365
FindCompIntfcs method 612
FindCRefByName method 1623
FindCRefByURL method 1624
FindCRefForURL method 1625
FindCRefLinkByName method 1627
FindExpression method 1929
FindFolderByName method 1627
FindKeyInfoCollection property 633
FindLeaf method 2364
Find method
Array class 266
Component Interface class 627
Mobile class 1400
FindNode method
Tree class 2365
XmlNode class 2565
FindNodes method 2566
FindPgltByName method 1629
FindPortalRegistries method 1611
FindQueriesDateRange method 1911
FindQueries method 1910
FindQueryDBRecords method 1910
FindRoot method 2366
FindRowNum method 1568
FirstChildLeaf property 2348
FirstChildNode property 2348
First method
AssignedPagelet collection 1751
Attribute collection 1708
AvailableCategory collection 1756
AvailablePagelet collection 1761
CompIntfPropInfoCollection collection 647
Component Interface collection 614
Content Reference collection 1701
DynamicCategory collection 1764
Favorite collection 1805
Folder collection 1679
Link collection 1735
Node collection 1649
NodeTemplate collection 1660
PageletCategory collection 1774
Pagelet collection 1786
ParseResult collection 2482
PermissionValue collection 1714
Portal collection 1656
PortalRegistry collection 1643
PSMessages collection 2192
Query collection 1919
QueryCriteria collection 1995
QueryDBRecord collection 2042
QueryDBRecordField collection 2048
QueryExpression collection 2015
QueryField collection 1980
QueryList class 2024
Query Metadata collection 2032
QueryPrompt collection 2058
QueryRecord collection 1974
QueryRecordHierarchy collection 2027
QuerySelect collection 1957
RemoteNode collection 1652
RolePermissionValue collection 1718
SearchField collection 2489
SearchIndexes collection 2492
Search Language collection 2506
SearchRecordField collection 2501
SearchResult collection 2485
Search Schedule collection 2510
Search Start Options collection 2521
SelectedPagelet collection 1799
TabDefinition collection 1746
UserTab collection 1794
First property 2322
FirstStep method 722
FirstUrl property
FeedDoc class 913
Flag property
ColumnNumber class 1988
QueryDBRecordField class 2054
flat table methods 305
Flattening 264
Flush method 2140
FlushRow method 2142
Folder class
adding folders 1811
list of methods, properties, returns 2815
methods 1663
properties *See Also* Folder class properties
understanding 1663

Folder class methods 1663
Folder class properties
  AbnContentProvider 1664
  AbnDataSource 1665
  AbnPeopleCode 1665
  Attributes 1666
  Author 1666
  AuthorAccess 1666
  Authorized 1667
  CascadedPermissions 1667
  CascadedRolePermissions 1668
  ContentRefs 1668
  CreationDate 1668
  DefaultChartNavigation 1669
  Description 1669
  Folders 1669
  IsMobile 1670
  IsVisible 1670
  Label 1670
  Name 1671
  OwnerId 1671
  ParentName 1671
  Path 1671
  Permissions 1672
  Product 1672
  PublicAccess 1672
  RolePermissions 1673
  SequenceNumber 1673
  TreeEffectiveDate 1673
  TreeName 1674
  TreeSetId 1675
  TreeStructureName 1676
  TreeUserKeyValue 1677
  ValidFrom 1678
  ValidTo 1678
Folder collection
  list of methods, properties, returns 2817
  methods *See Also* Folder collection methods
  properties 1682
  understanding 1678
Folder collection methods
  DeleteItem 1679
  First 1679
  InsertItem 1680
  ItemByName 1681
  Next 1681
Folder collection properties 1682
FolderName property 2611
  report class 2616
folder navigation
  FolderNavObject property 1640
  IsFolderNavigation property 1641
FolderNavObject property 1640
Folder property 1949
folders
  adding 1811
  collection *See Also* Folder collection
  deleting 1609
  FindFolderByName method 1627, 1628
  Folder class *See Also* Folder class
  Folder collection 1678
  Folder property 1949
  objects 1663
  PortalFolder property 1872
  ReportFolder property 1831
  RootFolder property 1642
  SetItemFolder method 1858
  understanding 1588
Folders property 1669
FontName property 2628
FontSize property 2629
FormatBinaryResultString method 1930
FormatLength property 952
Format property
  ColumnNumber class 1988
  CompIntfPropInfoCollection objects 649
  QueryDBRecordField class 2055
FormatResultString method 1931
FormattedValue property 953
FormatType property 107
freeze column mode 161
From property
  MCFEmail class 1128
  MCFOutboundEmail 1194
FSOpts property 2495
FullContent property
  FeedEntry class 927
FullURI property 1069
Func class 1091
FunctionCall class
  AddArgument method 147
  creating 143
  GenerateRule method 147
  GetArguments method 148
  methods 147
  Name property 149
  properties 149
  Type property 149
functions *See* PeopleCode functions
  CommitWork 1490
  CreateOptInterface 1510, 1543
  DeleteOptProbInst 1494, 1495, 1511
  DoSaveNow 1490
  GetOptEngine 1492, 1494, 1513
  GetOptProbInstList 1514
  InsertOptProbInst 1495, 1515
  IsValidOptProbInst 1517
  MessageBox 1496
  sending optimization status 1501
  WinMessage 1496
FuturePublicationDateTime property 1365

G

Gantt chart
  creating 403
  example 585
  methods 477
  properties 506
  specifying Date Time Axis formats 404
  using Gantt Glyphs 405
Gantt class
  handling errors 406
  list of methods 2674
  list of properties 2675
  objects 406
  properties *See Also* Gantt class properties
  understanding 400
Gantt class methods 477, 483
  Refresh method 477
  Reset method 478
  SetActualEndDate 478
  SetActualStartDate 479
  SetActualTaskBarColor 480

SetChartArea 480
SetDayFormat 481
SetHourFormat 482
SetMinuteFormat 484
SetMonthFormat 485
SetPlannedEndDate 486
SetPlannedStartDate 486
SetPlannedTaskBarColor 487
SetSecondFormat 488
SetTableXScrollbar 489
SetTaskAppData 489
SetTaskAppDataTitles 490
SetTaskBarURL 491
SetTaskData 492
SetTaskDependencyChildID 492
SetTaskDependencyData 493
SetTaskDependencyParentID 494
SetTaskDependencyType 494
SetTaskDependencyURL 496
SetTaskDrill 497
SetTaskExpanded 497
SetTaskHints 498
SetTaskID 499
SetTaskLabel 499
SetTaskLevel 500
SetTaskMilestone 501
SetTaskName 502
SetTaskProgress 503
SetTaskProgressBarColor 503
SetWBSNumbering 504
SetYearFormat 505
Gantt class properties
AxisEndDateTime 506
DataEndDateTime 507
DataStartDateTime 507
DataStartRow 507
DataWidth 507
GridLines 508
GridLineType 509
HasLegend 509
Height 510
ImageMap 510
InteractiveMove 511
InteractiveProgress 511
InteractiveStart 511
IsDrillable 511
IsPlainImage 510, 512
LegendPosition 513
LegendStyle 513
MainTitle 514
MainTitleStyle 514
PixelsPerRow 514
RevertToPre850 515
ShowTaskLabels 515
Style 516
StyleSheet 516
TaskDependencyLineType 516
TaskMilestoneGlyph 517
TaskTitle 517
Width 518
XAxisPosition 518
YAxisPosition 518
garbage collection
Java programs 1097
GenABNMenuElement method
Leaf class 2305
Node class 2329
GenBreadCrumbs method
Node class 2330
Generate functions 1057
generateHTML method
MCFIMSingleButton class 1240
generateJavaScript method
MCFIMSingleButton class 1242
Generate method 243
GenerateRule method
assignment class 132
comparison class 134
constant class 137
cube class 140
FunctionCall class 147
MemberReference class 149
Operation class 151
RuleDefn class 130
variable class 154
Generator property
FeedDoc class 914
generic base classes 223
genFeedUrl method
FeedFactory class 802
GenFormattedXmlString method 2544
genNameSpaceID method
Utility class 859
GenRelatedActions method
Node class 2331
genUniqueFeedId method
FeedFactory class 803
GenXmlFile method
XmlDoc class 2545
GenXMLPartString method 1271
GenXmlString method
Document class 730
XmlDoc class 2546
XmlNode class 2567
GenXMLString method
Incoming Business Interlink class 363
Message class 1271
GetAbsoluteContentURL method 1630
GetActualRemoteNodes method 1612
GetAdditionalUserInfo method 1236
GetAEAttributeName method
IBInfo class 1347
GetAEAttributeValue method
IBInfo class 1347
GetAggregateMapping method 110
getAllFileNames method 1882
getAllPagedFeedLinks method
FeedFactory class 804
GetAnalyticModel method 20
GetAndExpandTemplate method 1461
GetArchData method 1316
GetArchIBInfoData method 1317
GetArguments method 148
GetASyncBPELProcessInstanceUrl method 292
GetAttachmentContentID method
IBInfo class 1348
GetAttachmentProperty method
IBInfo class 1349
GetAttachments method 1148
getAttributeById method
DataSource class 818
getAttribute method
Feed class 779
GetAttributeName method
IBInfo class 1351
Incoming Business Interlink class 364

XmlNode class 2568
GetAttributeValue method
IBInfo class 1352
Incoming Business Interlink class 365
XmlNode class 2569
GetAuxFlag method 940
GetBodyPart method 1176
GetBPELConsoleUrl method 298
GetBPELDomain method 299
GetBPELProcessBrowserUrl method 293
getCategory method
FeedFactory class 805
GetCauses method 100
GetCDataValue method 2569
GetCDataValues method 2570
GetCellProperties method 30
GetChildNode method 2571
GetCircularDeps method 101
GetColumn method
AnalyticGrid class 163
Grid class 1039
GetCompIntfc method 613, 1398
GetConnectorPropertiesName method 1378
GetConnectorPropertiesType method 1379
GetConnectorPropertiesValue method 1380
GetContainerAttributeName method
IBInfo class 1352
GetContainerAttributeValue method
IBInfo class 1353
GetContentBody method 1062
GetContentString method 1272
GetContentType method 1177
getContentUrl method
DataSource class 819
GetCookie method 1075
GetCookieNames method
Request class 1063
Response class 1076
GetCookieValue method 1064
GetCount method
Business Interlink class 334
MCFGetMail class 1132
MCFMultiPart class 1177
GetCubeCollection
analytic grids 164
GetCubeCollection method
AnalyticModel class 30
cube collection 75
GetCubeCollectionNames method 76
GetCube method 74
GetCubeNames method
AnalyticModelDefn class 77
CubeCollectionDefn class 111
GetCurrEffRow method 2143
GetData method 40
getDataSecurity method
DataSource class 819
getDataSource method
FeedFactory class 806
GetDateArray method 1523
GetDate method 1522
GetDateTimeArray method 1525
GetDateTime method 1524
GetDefaultHPTabOID method 1630
GetDimensionAggregate method 101
GetDimension method 77
GetDimensionNames method
AnalyticModelDefn method 78
CubeCollectionDefn class 111
CubeDefn class 103
ExplicitDimensionSet class 97
GetDimFilter method 41
GetDimSort method
CubeCollection class 42
CubeCollectionDefn class 112
GetDoc method 335
GetDocumentKey method
Document class 731
GetDocument method 1273
GetDummyRows property 633
GetEffectiveItem method
description 638
using 610
GetEffectiveItemNum method 639
GetEffects method 102
GetElement method 2572
Document class 732
GetElementsByTagName method
XmlDoc class 2546
XmlNode class 2574
GetElementsByTagNameNS method 2575
GetElements method 2573
GetEmailCount method 1133
getEntry method
FeedDoc class 908
GetEnumName method
Primitive class 741
GetErrorImageName method 1236
GetErrorMsgParam method
MCFMailUtil class 1167
MCFOutboundEmail class 1181
getExceptionText method
Utility class 859
GetExplicitDimensionSet method 79
GetExplicitDimensionSetNames method 80
getFeedDoc method
FeedFactory class 809
Utility class 860
getFeedLink method
FeedFactory class 810
getFeedLinks method
FeedFactory class 810
getFeed method
FeedFactory class 807
getFeedMimeType method
Utility class 861
GetFieldCount method 338
GetFieldMapping method 113
GetField method
QueryRecord class 1977
Record class 2089
getFieldTranslates method
Utility class 861
GetFieldType method 339
GetFieldValue method 340
GetFilter method 114
GetFirstUserSortedRow method 2143
GetFrom method 1149
GetHeaderCount method
MCFBodyPart class 1118
MCFOutboundEmail class 1182
GetHeader method
MCFBodyPart class 1118
MCFOutboundEmail class 1181
Request class 1064
Response class 1076

GetHeaderName method
MCFBodyPart class 1119
MCFOutboundEmail class 1183
GetHeaderNames method
MCFBodyPart class 1120
MCFOutboundEmail class 1183
Request class 1064
Response class 1076
GetHeaderValues method
MCFBodyPart class 1120
MCFOutboundEmail class 1184
GetHelpURL method 1065
GetHistoryItems property 634
GetIBInfoData method 1318
GetIBTransactionIDforAE method 1319
GetImageFormat method 1985, 2052
GetImageURL method 1077
GetIndexes method 140
GetInputDocs method 341
GetItem method
Collection class 756
GetJavaScriptURL method 1077
GetKeyInfoCollection property 634
Get keyword 215
Get language constructs 227
GetLastUserSortedRow method 2144
GetLaunchURL method 1238
GetLayout method 43
GetListViewAttrs method 1426
GetLocalNode method 1613
GetLogicalQueue method 2462
GetLongLabel method 941
GetMembers method 31
GetMessageErrors method 1321
GetMessageId method 300
GetMessage method 1319
Get method
AnalyticModelDefn class 74
analytic type definition 178
Array class 267
Component Interface class 627
Mobile class 1401
ReportDefn class 2602
RowsetCache class 2174
GetModel method 178
GetModelNames method 179
GetMsgSchema method 1321
GetName method 1402
GetNewEffRow method 2145
GetNextDoc method 342
GetNextEffRow method 2114
GetNode method
IBUtil class 300
Incoming Business Interlink class 366
GetNodes method 1613
getNodeValue method
Utility class 862
GetNumberArray method 1527
GetNumber method 1526
GetNumberofAEAttributes method
IBInfo class 1354
GetNumberOfAttributes method
IBInfo class 1354
GetNumberOfConnectorProperties method 1380
GetNumberOfContainerAttributes method
IBInfo class 1355
GetNumberOfQueryStringArgs method 1381
GetNumberOfRoutings method 301
GetOffLineImageName method 1237
GetOnlineImageName method 1237
GetOperationType method 293
GetOptEngine function 1492, 1494, 1513
GetOptProbInstList function 1514
GetOrganizer method 80
GetOrganizerNames method 81
GetOutDestFormatString method 2603
GetOutputDocs method 344
getParameterById method
DataSource class 820
getParameterDetail method
DataSource class 821
GetParameter method 1065
GetParameterNames method 1065
GetParameterValues method 1066
GetParent method 1403
Collection class 756
Compound class 748
Primitive class 742
GetParmDateArray method 1545
GetParmDate method 1544
GetParmDateTimeArray method 1546
GetParmDateTime method 1546
GetParmIntArray method 1549
GetParmInt method 1548
GetParmName method 2227
GetParmNumberArray method 1548
GetParmNumber method 1547
GetParmStringArray method 1550
GetParmString method 1550
GetParmTimeArray method 1552
GetParmTime method 1551
GetParmValue method 2228
GetPartAliasName method 1274
GetPartName method 1274
GetPartNames method 127
GetPartRowset method 1275
GetParts method 1149
GetPartVersion method 1276
GetPartXMLDoc method 1276
GetPath method
Collection class 757
Compound class 749
Primitive class 742
GetPeerDefaultAttributesByName method 1404
GetPersistAggregate method 114
GetPortalRegistry method 1614
GetPosition method 977, 986
GetPreviousDoc method 345
GetPriorEffRow method 2114
GetPropertyAttrsByName method 1404
GetPropertyByIndex method
Compound class 749
GetPropertyByName method
Component Interface class 628
Compound class 750
Mobile class 1405
GetPropertyInfoByName method 1406
GetProperty method 2636
GetPSQueryPromptRecord method 2604
GetQualifiedURL method 1631
GetQuery method 1912
GetQueryScopeByName method
UTILITY class 712
GetQuerySecurityProfile method 1913
GetQueryStringArgName method 1381
GetQueryStringArgValue method 1382

GetQueryString method 1277
GetRecord method
    AnalyticModelDefn class 180
    Row class 2115
GetRecordNames method 180
getRelatedFeedsHoverMenu method
    FeedFactory class 811
GetRelated method 942
getRelativePageLinkForFeed method
    FeedFactory class 812
GetRemoteNodes method 1615
GetReportList method 2620
GetResponseStatus method 1469
GetRowCount method 44
GetRow method 2146
GetRowset method
    Document class 733
    Message class 1278
    Row class 2117
GetRuleExpressions method 142
GetRule method
    CubeDefn class 103
    UserFunctionDefn class 123
GetSampleXMLString method
    CONQRSMGR class 675
GetSchema method
    Document class 734
GetSearchIndexes method 2472
GetSearchQuery method
    PortalRegistry class 2474
    Session class 2473
GetSegment method 1280
GetSelectedField method 184
GetSender method 1150
getSettingById method
    DataSource class 821
getSettingDetail method
    DataSource class 822
GetShortLabel method 943
GetSlice method 44
GetSolutionDetail method 1571
GetSolution method 1569
GetStatus method
    Business Interlink class 347
    IBUtil class 301
    MCFIMInfo class 1239
GetStringArray method 1529
GetString file class method 988
GetString method 1528
GetStyleSheetURL method 1078
GetSubstitution method 765
GetSyncBPELProcessInstanceUrl method 294
GetSyncIBInfoData method 1322
GetSyncLogData method 1323
GetTimeArray method 1532
GetTimeDiff method 2463
GetTime method 1531
getting
    RuleExpression 142
GetTopParent method 1407
GetTraceLevel method 1533
GetTransactionIDforAE method
    IBInfo class 1356
GetTransactionId method 302
GetTree method 2301
GetTree methods 33
GetTreePromptCount method
    Query class 1932
GetTreeStructure method 2301
GetUniqueKey method
    Compound class 751
GetUnknownImageName method 1238
GetUnparsedHeaders method 1121
getUserDateFormat method
    Utility class 863
getUserDatetimeFormat method
    Utility class 863
GetUserFunction method 82
GetUserFunctionNames method 82
getUserInfo method
    Utility class 864
GetValue method 349
GetXmlDoc method 1280
global
    declaring Business Interlink objects 310
    variables in application classes
        See Also global variables
global variables
    availability with PortalRegistry classes 1610
    going out of scope with SQL objects 2247
    using in application classes 217
GlobList property
    Search FS Options class 2518
    Search HTTP Options class 2515
GlobListType property
    Search FS Options class 2519
    Search HTTP Options class 2515
GoToStep method 723
GrantPermissionForComponent method 1631
GrantPermissionForScript method 1633
Greeting property 1790
Grid class
    functions, methods, properties, returns 2720
    methods 1038
    properties 1044
    shortcut considerations 1036
    understanding 1035
    using in PeopleCode 1036
grid classes
    functions, methods, properties, returns 2720
Grid class methods
    EnableColumns 1038
    GetColumn 1039
    LabelColumns 1041
    SetProperties 1042
    ShowColumns 1043
Grid class properties
    gridcolumn 1044
    Label 1045
    SummaryText 1045
GridColumn class
    list of properties, returns 2721
    properties
        See Also GridColumn class properties
    understanding 1045
GridColumn class properties
    Enabled 1046
    Label 1047
    Name 1047
    Visible 1048
GridColumn objects
    declaring/scope 1037
    understanding 1045
gridcolumn property 1044
grid columns
    class *See Also* GridColumn class

gridcolumn property 1044
objects *See Also* GridColumn objects
GridLines property 455, 508
GridLineType property 509
grid objects
declaring/scope 1037
understanding 1036
grids *See Also* analytic grids
columns *See Also* grid columns
Grid class *See Also* Grid class
GridLines property 455, 508
GridLineType property 509
objects *See Also* grid objects
understanding 1035
user sorted 2141
GUID property 1298
FeedEntry class 928

H

HasAdminParams property
Feed class 794
HasChildLeaves property 2349
HasChildNodes property 2349
HasChildren property 2349
HasDefaultValue property 1420
HasDetailRanges property 2389
HasLegend property 456, 509, 530, 548
HasLockedBranches property 2389
HasNextSib property
Leaf class 2316
Node class 2349
HasParameters property
DataSource class 828
HasPrevSib property
Leaf class 2316
Node class 2350
HasSettings property
DataSource class 828
HasUserParams property
Feed class 794
HavingCriteria property 1971
HeaderNode property 2233
headers
comments about the class 222
comments for methods 221
SOAPDoc objects 2214
SOAP header examples 2236
HeadingText property
ColumnNumber class 1989
QueryPrompt class 2064
HeadingType property
QueryField class 1989
QueryPrompt class 2064
HeadingUniqueFieldName property 1989
Height property 456, 510, 530, 548
HelpID property
Pagelet class 1781
TabDefinition class 1741
HelpLink property 1584
HideAllRows method 2147
hideDelay property
MCFIMSingleButton class 1244
hints, data 439
HitCount property 2478
Homepage property 1640
homepages
accessing the Homepage tab 1737
DeleteHomepage method 1622
deleting templates 1609
Homepage property 1640
understanding 1590
UserHomepage class
See Also UserHomepage
HostNodeName property 1655
HoverText property 954
HtmlText property
Content Reference class 1690
TabDefinition class 1741
HTTP
HTTPMethod property 1069
HTTPOpts property 2495
Search HTTP Options
See Also Search HTTP Options
HTTPMethod property 1069
HTTPOpts property 2495
HTTPSessionId property 1365
httpStringToDatetime method
Utility class 864

I

IBConnectorInfo collection
list of methods, properties, returns 2746
methods
See Also IBConnectorInfo collection methods
objects 1374
properties 1382
understanding 1374
IBConnectorInfo collection methods
AddQueryStringArg method 1375
ClearConnectorProperties method 1376
ClearQueryStringArgs method 1377
dConnectorProperties method 1374
DeleteConnectorProperties method 1377
DeleteQueryStringArg method 1378
GetConnectorPropertiesName method 1378
GetConnectorPropertiesType method 1379
GetConnectorPropertiesValue method 1380
GetNumberOfConnectorProperties method 1380
GetNumberOfQueryStringArgs method 1381
GetQueryStringArgName method 1381
GetQueryStringArgValue method 1382
IBConnectorInfo property 1366
IBException class
instantiating 1299
IBException property 1299
IBInfo
class *See Also* IBInfo class
objects 1336
IBInfo class
adding attachments 1338
Application Engine handler 1336
clearing attachments 1342
content ID 1348
deleting attachments 1344
getting attachment properties 1349
list of methods, properties, returns 2743
methods *See Also* IBInfo class methods
properties *See Also* IBInfo class properties

setting attachment properties 1360
understanding 1336
IBInfo class methods
AddAEAttribute 1336
AddAttachment 1338
AddAttribute 1339
AddContainerAttribute 1340
ClearAEAttributes 1341
ClearAttachments 1342
ClearAttributes 1342
ClearContainerAttributes 1343
DeleteAEAttribute 1344
DeleteAttachment 1344
DeleteAttribute 1345
DeleteContainerAttribute 1346
GetAEAttributeName 1347
GetAEAttributeValue 1347
GetAttachmentContentID 1348
GetAttachmentProperty 1349
GetAttributeName 1351
GetAttributeValue 1352
GetContainerAttributeName 1352
GetContainerAttributeValue 1353
GetNumberofAEAttributes 1354
GetNumberOfAttributes 1354
GetNumberOfContainerAttributes 1355
GetTransactionIDforAE 1356
InsertAEResponseAttributes 1357
LoadConnectorProp 1357
LoadConnectorPropFromNode 1358
LoadConnectorPropFromRouting 1359
LoadConnectorPropFromTrx 1360
SetAttachmentProperty 1360
IBInfo class properties
AppServerDomain 1362
CompressionOverride 1362
ConnectorOverride 1363
ConversationID 1363
DeliveryMode 1363
DestinationNode 1364
ExternalMessageID 1364
ExternalOperationName 1364
ExternalUserName 1364
ExternalUserPassword 1365
FinalDestinationNode 1365
FuturePublicationDateTime 1365
HTTPSessionId 1365
IBConnectorInfo 1366
InReplyToID 1366
MessageChannel 1366
MessageName 1366
MessageQueue 1367
MessageType 1367
MessageVersion 1367
NodeDN 1368
NonRepudiationID 1368
NumberOfAttachments 1368
OperationType 1368
OperationVersion 1369
OrigNode 1369
OrigProcess 1369
OrigTimeStamp 1369
OrigUser 1370
PublicationID 1370
RequestingNodeDescription 1370
RequestingNodeName 1370
ResponseAsAttachment 1370
SegmentsUnOrder 1371
SourceNode 1371
SyncServiceTimeout 1371
TransactionID 1372
UserName 1372
VisitedNodes 1372
WSA_Action 1373
WSA_FaultTo 1373
WSA_MessageID 1373
WSA_ReplyTo 1373
WSA_To 1374
IBInfo objects 1336
IBInfo property 1299
IBNode property
MCFGetMail class 1144
MCFInboundEmail class 1154
IBOperationName property
Feed class 795
IBOperations property
DataSource class 828
IBSOTYPE_ASYNC property
Utility class 886
IBSOTYPE_SYNC property
Utility class 886
IBSOTYPE_UNKNOWN property
Utility class 886
IBUtil class
methods 298
understanding 298
IBUtil class methods
GetBPELConsoleUrl 298
GetBPELDomain 299
GetmessageId 300
GetNode 300
GetNumberOfRoutings 301
GetStatus 301
GetTransactionId 302
IBUtil constructor 290
IconOnlySelectedQuadrantStyle property 549
ICONURL_FEED_A property
Utility class 886
ICONURL_FEED_IA property
Utility class 887
ID property 25
DataSource class 828
DataSourceParameter class 838
DataSourceParameterValue class 844
DataSourceSetting class 850
Feed class 795
FeedDoc class 914
FeedEntry class 928
IgnoreInvalidId property 1010
IgnoreRelFldOutput property
CONQRSMGR class 685
ImageFileLowerLeftX property 2640
ImageFileLowerLeftY property 2640
ImageFile property 2639
ImageFileUpperRightX property 2640
ImageFileUpperRightY property 2640
ImageHeight property 531, 532
ImageLocation property 531
ImageMap property 457, 510
ImageName property 2316
imagesLocation property 1174
implementing
feed request handlers 930
Importance property
MCFEmail class 1129
MCFOutboundEmail 1194

Import function 228
importing
  application class names 209
  BPEL classes 288
  class names in general 224
  connected query classes 672
  feed classes 775
  Import function 228
  mail classes 1102
  MCFMailStore class 1157
  Notification class 1453
  Notification classes 1446
  PrcsApi class 1881
  universal queue classes 2423
import statements
  CONQRSMGR class 672
  DataSource class 815
  DataSourceParameter class 831
  DataSourceSetting class 846
  Feed class 776
  FeedDoc class 903
  FeedEntry class 918
  FeedFactory class 799
  SEC_PROFILE class 709
  Utility class 854
  UTILITY class 710
Inactive property
  AnalyticGrid class 167
InBoundPublish method 1281, 1324
Incoming Business Interlink class
  methods
    See Also Incoming Business Interlink class methods
  properties
    See Also Incoming Business Interlink properties, Incoming Business Interlink class properties
Incoming Business Interlink class methods
  AddAttribute 357
  AddComment 358
  AddProcessInstruction 359
  AddText 361
  CreateElement 362
  GenXMLString 363
  GetAttributeName 364
  GetAttributeValue 365
  GetNode 366
  ParseXMLString 367
  using 309
Incoming Business Interlink class properties
  AttributeCount 368
  ChildNodeCount 369
  NodeName 370
  NodeType 370
  NodeValue 371
  using 309
INCREMENTALOPTION_NO property
  Utility class 887
INCREMENTALOPTION_YES property
  Utility class 887
IncrementalView property 2498
Indexes property 2478
indexing
  BuildSearchIndex method 2474
  GetSearchIndexes method 2472
  Indexes property 2478
  IndexName property 2479
  search indexes *See Also* search indexes
IndexName property 2479
Index property 2590
IndirectionMethod property 2411
InError property 1437
InitExisting property
  CONQRS_CONST class 699
Init method 1552
InitNew property
  CONQRS_CONST class 700
InputRowset method 305, 350
InReplyToID property 1366
InsertAEResponseAttributes method
  IBInfo class 1357
InsertCDataSection method 2576
InsertChildLeaf method 2331
InsertChildNode method 2332
InsertChildRecord method 2333
InsertComment method 2577
InsertDynChildLeaf method 2334
InsertDynSib method 2305
InsertElement method 2579
InsertElementNS method 2580
InsertEnabled property 2165
InsertEntityReference method 2581
InsertItem method
  AssignedPagelet collection 1752
  Attribute collection 1708
  Collection class 757
  Component Interface class 610
  Content Reference collection 1702
  Data collection 640, 1429
  DynamicCategory collection 1765
  Favorite collection 1805
  Folder collection 1680
  NodeTemplate collection 1661
  PageletCategory collection 1775
  Pagelet collection 1786
  PermissionValue collection 1714
  RolePermissionValue collection 1719
  Search Language collection 2507
  SearchRecordField collection 2502
  Search Schedule collection 2511
  Search Start Options collection 2522
  SelectedPagelet collection 1800
  TabDefinition collection 1747
  UserTab collection 1794
Insert method 2090
InsertNode method 2582
InsertOptProbInst function 1495, 1515
InsertProcessInstruction method 2584
InsertRoot method 2367
InsertRow method 2148
InsertSib method
  Leaf class 2306
  Node class 2335
InsertSibRecord method 2336
InsertText method 2585
insertXMPPServerUserData method
  MCFIMSingleButton class 1242
INSTALLATION table 973
instanceid property 1477
InstanceID property
  WSWorklistEntry class 1484
instance variables
  creating public properties 213
  declaring private 198
  declaring private in application classes 217
  overriding properties 214

passing as function arguments 207
running constructors 204
instant messaging
adding users 1235
checking user status 1235
MCF IM classes *See Also* MCF IM classes
removing users 1239
supported networks 1233
Instruction property 1464
instselectedddttm property 1477
inststatus property 1478
insttimeoutddttm property 1478
instworkedddttm property 1478
IntBroker
class *See Also* IntBroker class
objects 1312
IntBroker class
list of methods and returns 2741
list of variables and returns 2741
methods *See Also* IntBroker class methods
understanding 1312
IntBroker class methods
Cancel 1312
ConnectorRequest 1314
ConnectorRequestUrl 1315
DeleteOrphanedSegments 1316
GetArchData 1316
GetArchIBInfoData 1317
GetIBInfodata 1318
GetIBTransactionIDforAE 1319
GetMessage 1319
GetMessageErrors 1321
GetMsgSchema 1321
GetSyncIBInfoData 1322
GetSyncLogData 1323
InBoundPublish 1324
IsOperationActive 1325
MsgSchemaExists 1326
Publish 1326
Resubmit 1328
SetMessageError 1329
SetStatus 1331
SwitchAsyncEventUserContext method 1332
SyncRequest method 1332
Update 1334
UpdateXmlDoc 1335
IntBroker objects 1312
Inteface language construct 229
integration
API Repository 234
Business Interlink class 303
Component Interface classes 599
Integration Broker 1114
Integration Broker 1114
Integration Broker BPEL classes
See BPEL classes
InteractiveEnd property 510
InteractiveMode property 635
InteractiveMove property 511
InteractiveProgress property 511
InteractiveStart property 511
interfaces
component *See Also* Component Interface
declaring public external 193
implementing 203
PeopleCode 202
understanding 203
InternalURL property 2451
InvalidAddresses property 1194
IsActive property
Message class 1299
PhysicalQueue class 2451
IsAltKey property 954
IsAttachment property
MCFBodyPart class 1126
SeachRecordField class 2504
IsAuditFieldAdd property 954
IsAuditFieldChg property 955
IsAuditFieldDel property 955
IsAuthenticationReqd property
MCFOutboundEmail class 1195
SMTPSession class 1210
IsAutoUpdate property 955
IsBranched property
Node class 2350
Tree class 2389
IsBulkLoadTruncation property 1300
IsChanged property
Collection class 759
Compound class 752
CONQRSMGR class 686
Field class 955
Leaf class 2316
Node class 2350
Primitive class 743
Record class 2105
Row class 2120
Tree class 2390
IsCollection property
CompIntfPropInfoCollection objects 650
PropertyInfo object 1420
IsCreatedViaWebService 1478
iScript classes
Cookie *See Also* Cookie class
creating web libraries 1053
Request *See Also* Request class
Response *See Also* Response class
returns 2721
scope 1059
system variables 2721
understanding 1051
URLs vs. URIs 1053
iScripts
adding security 1054
avoiding Java applets/browser plug-ins 1056
classes *See Also* iScript classes
creating 1053
creating charts 596
generating charts 382
handling errors 1059
objects 1060
understanding 1051
using styles/stylesheets 1056
viewing 1056
when to use 1055
isCurrentUserAdmin method
DataSource class 823
isCurrentUserAuthorized method
DataSource class 823
IsCut property
Leaf class 2317
Node class 2350
IsDateRangeEdit property 956
IsDebugMode property
CONQRSMGR class 686
IsDefault property 1647

IsDeleted property
- Leaf class 2317
- Node class 2350
- Record class 2105
- Row class 2121

IsDelta property 1300
IsDescKey property 956
IsDomainNameValid method 1168
IsDomainRestricted property 2524
IsDragable property 549
IsDrillable property 457, 511
IsDuplKey property 956
IsEditError property
- Message class 1301
- Record class 2106
- Row class 2121
- Rowset class 2166

IsEditTable property 956
IsEditXlat property 957
IsEmailServerAvailable method 1169
IsEmpty property 1302
IsError property 1010
IsFolderNavigation property 1641
IsFromSearchField property 957
IsHideActionBar property 1741
IsHideMinimize property 1781
IsHostRestricted property 2524
IsInBuf property 957
IsInitialized property
- Collection class 760
- Compound class 752
- Primitive class 744

IsInit property
- CONQRSMGR class 686

IsInserted property
- Leaf class 2317
- Node class 2351

IsKey property 958
IsLayoutLocked property 1741
IsListItem property 958
IsLocal property
- Message class 1302
- Node class 1647
- Portal class 1655

IsMinimized property 1797
IsMobile property
- Content Reference class 1690
- ContentReference links 1727
- Folder class 1670

IsModelActive OptInterface class methods 1573
IsNewAndNeverSaved method 1408
IsNewFileId property 1011
IsNew method 1407
IsNew property 2122
IsNotUsed property 958
IsNull property
- XmlDoc class 2552
- XmlNode class 2590

IsOkToSendPartial property 1195
IsOpen property
- AESection class 11
- File class 1012
- SQL class 2255
- Tree class 2390

IsOperationActive method 1325
IsOperationActive property 1303
IsParts property 1303
IsPartsStructured property 1303
IsPeerReference property 1421
IsPlainImage property 458, 512
IsPrompt property 2026
IsPublic property
- CONQRSMGR class 687

IsQueryTree property 2390
IsReadOnly property 650
IsReferenceUnresolved property 2279
IsRefToParent property 1421
IsRenameable property 1742
IsRequest property 1304
IsRequired property 959
- Collection class 760
- Compound class 752
- Primitive class 744

IsReturnReceiptReqd property 1195
IsRichTextEnabled property 959
IsRoot property 2351
IsSameAs method 1409
IsSearchItem property 959
IsSet method
- ListViewAttrs class 1431
- PeerDefaultAttributes class 1439
- PropertyAttrs class 1435

IsSimpleApplicationProperty property 1421
IsSourceNodeExternal property 1304
IsStructure property 1304
IsSystem property 959
IsSystemProperty property 1422
IsThroughSearchField property 959
IsUseDefaultLabel property 960
IsUserSorted method 2149
IsValidOptProbInst function 1517
IsValid property
- AnalyticModelDefn class 91
- Tree class 2391

IsVerityField property 2505
IsVersionChanged property 2391
IsVirtual property 107
IsVisible property
- Content Reference class 1690
- ContentReference links 1727
- Folder class 1670

IsWholeTree property 2391
IsWordIndex property 2505
IsXlatExpression property 2020
IsYAxisInteger property 458
IsYesNo property 960
ItemByAlias method 1975
ItemByExpNum method 1982
ItemByKeys method 641
ItemByLabel method 1806
ItemByNameAndAlias method 1981
ItemByName method
- AssignedPagelet collection 1753
- Attribute collection 1709
- AvailableCategory collection 1757
- AvailablePagelet collection 1761
- ClassInfo collection 248
- Content Reference collection 1703
- DynamicCategory collection 1765
- Folder collection 1681
- MethodInfo collection 251
- Namespaces collection 245
- Node collection 1650
- NodeTemplate collection 1661
- PageletCategory collection 1775
- Pagelet collection 1787

PermissionValue collection 1715
Portal collection 1656
PropertyInfo collection 254
Query collection 1920
QueryCriteria collection 1996
QueryDBRecord collection 2043
QueryDBRecordField collection 2049
QueryExpression collection 2017
Query Metadata collection 2033
QueryPrompt collection 2059
QueryRecordHierarchy collection 2028
RemoteNode collection 1652
RolePermissionValue collection 1719
SearchField collection 2489
SearchIndexes collection 2492
Search Language collection 2507
SearchRecordField collection 2502
Search Schedule collection 2511
Search Start Options collection 2522
SelectedPagelet collection 1800
TabDefinition collection 1747
UserTab collection 1795
ItemBySelNum method 1958
Item method
Bindings collection 242
CI collection 1424
ClassInfo collection 247
CompIntfPropInfoCollection collection 648
Component Interface collection 614
Data collection 641, 1427
Level collection 2321
MethodInfo collection 250
Namespaces collection 244
PortalRegistry collection 1644
PropertyInfo collection 254
PropertyInfoCollection class 1417
PSMessages collection 2193
Query collection 1920
QueryCriteria collection 1995
QueryDBRecord collection 2043
QueryDBRecordField collection 2049
QueryExpression collection 2016
QueryField collection 1981
QueryList class 2024
Query Metadata collection 2032
QueryPrompt collection 2059
QueryRecord collection 1974
QueryRecordHierarchy collection 2028
QuerySelect collection 1958
SearchResult collection 2485
ItemNum property 643

J

JAR files 1086
Java
applets 1056
calling application classes 1094
class *See Also* Java class
classes delivered with PeopleTools 1086
creating objects *See Also* Java objects
custom classes 1086
debugging 1098
functions *See Also* Java functions
JAR files 1086
Java Debugging Environment (JDB) 1098
Java Runtime Environment (JRE) 1086
managing states 1088
mapping PeopleCode types 1094
packages delivered with PeopleTools 1086
setting up the system for 1086
supported version 1086
third-party classes 1086
Java applets 1056
Java class
arrays 1097
calling PeopleCode 1090
copying data 1097
Func 1091
garbage collector 1097
getting system classes 1090
handling errors 1098
list of functions, methods, properties, returns 2725
Name 1091
SysCon 1091
SysVar 1091
understanding 1085
using PeopleCode functions 1096
Java Debugging Environment (JDB) 1098
Java functions
error handling 1098
understanding 1085
Java objects
creating 1089
declaring/scope 1099
Java Runtime Environment (JRE) 1086
JDB 1098
JobName property 1868
jobs
defining 1836
displaying items 1848
displaying status 1849
JobName property 1868
jobsets 1836
scheduling with item changes 1879
scheduling without item changes 1878
jobsets
displaying items 1848
displaying status 1849
scheduling 1855
understanding 1836
join2D method
Utility class 866
JoinAlias property 1978
JoinFieldName property 1978
join method
Utility class 865
Join method 268
joins
AllowAnyJoin property 2038
JoinAlias property 1978
JoinFieldName property 1978
JoinType property 1978
MaxJoins property 2041
JoinType property 1978
JRE 1086

K

KeyBranchName property 2392
KeyEffDt property 2392

KeyName property
  Tree class 2392
  Tree Structure class 2412
Key property
  CompIntfPropInfoCollection objects 650
  SearchResult class 2487
keys
  CompIntfPropInfoCollection Key property 650
  Get/Set keywords 215
  SearchResult class Key property 2487
  setting for a Component Interface 602
KeySetId property 2393
KeyUserKeyValue property 2393
KnowledgeBase property 2480

L

LabelColumns method
  Grid class 1041
LabelImage property 961
LabelLong property 651
Label property
  AnalyticGrid class 167
  AttributeValue class 1705
  Content Reference class 1691
  ContentReference links 1728
  Favorite class 1803
  Field class 960
  Folder class 1670
  Grid class 1045
  GridColumn class 1047
  ListViewAttrs class 1433
  PageletCategory class 1770
  Pagelet class 1781
  PeerDefaultAttributes class 1441
  PropertyAttrs class 1437
  TabDefinition class 1742
  UserTab class 1791
labels
  chart considerations 383
LabelShort property 651
LangCount property 2065
language_cd property 1456
LanguageCd property
  ProcessRequest class 1868
  Search Language class 2509
LanguageCD property
  RegionalSettings class 2201
language constructs, application class
  See application class language constructs
Language property
  Agent class 2438
  MCFInboundEmail class 1154
  NotificationAddress class 1460
  NotificationTemplate class 1465
  SearchQuery class 2480
  Task class 2458
languages
  Agent class Language property 2438
  application class language constructs *See Also* application class language constructs
  MapLanguageCd property 2509
  NotificationAddress class Language property 1460
  Notification class language_cd property 1456
  NotificationTemplate class Language property 1465
  ProcessRequest class LanguageCd property 1868
  RegionalSettings LanguageCD property 2201
  related languages for a Component Interface 606
  searches *See Also* Search Language
  SearchIndex class Language property 2495
  Search Language class LanguageCd property 2509
  SearchQuery class Language property 2480
  setting for email 1112
  setting for queries 1901
  Task class Language property 2458
Languages property 2495
LastChildLeaf property 2351
LastChildNode property 2352
LastExecDtTm property 2037
Last property 2322
LastPubDTTM property
  Feed class 795
LastSQLErrorCode property 1949
LastSyncDateTime property 2279
LastSyncRowCount property 2279
LastUpdatedBy property
  CONQRSMGR class 687
LastUpdateDTTM property
  CONQRSMGR class 687
LastUpdDttm property 1949
LastUpdDTTM property
  Feed class 795
LastUpdOprId property 1950
LastUpdOprID property
  Feed class 796
LastUrl property
  FeedDoc class 914
LaunchASyncBPELProcess method 295
LaunchSyncBPELProcess method 296
LayoutBehavior property
  AssignedPagelet class 1749
  AvailablePagelet class 1759
  TabDefinition class 1742
leaf
  ChildLeafCount property 2347
  class *See Also* Leaf class
  FindLeaf method 2364
  FirstChildLeaf property 2348
  InsertChildLeaf method 2331
  InsertDynChildLeaf method 2334
  LastChildLeaf property 2351
  LeafCount property 2393
  LeafExists method 2368
  LeafImageName property 2393
  LeafOnClipboard property 2394
  objects 2302
Leaf class
  list of methods, properties, returns 2857
  methods *See Also* Leaf class methods
  properties *See Also* Leaf class properties
  understanding 2302
Leaf class methods
  Cut 2303
  Delete 2303
  DeleteByRange 2304
  GenABNMenuElement 2305

InsertDynSib 2305
InsertSib 2306
LoadABNChart 2307
LoadABNChartOrdered 2308
MoveAsChild 2309
MoveAsChildByName 2310
MoveAsSib 2311
MoveAsSibByRange 2312
PasteSib 2312
RefreshDescription 2313
UpdateRanges 2314
Leaf class properties
Description 2315
DisplayLevelNumber 2315
Dynamic 2315
HasNextSib 2316
HasPrevSib 2316
ImageName 2316
IsChanged 2316
IsCut 2317
IsDeleted 2317
IsInserted 2317
NextSib 2317
Parent 2318
PrevSib 2318
RangeFrom 2318
RangeTo 2319
TreeBranchName 2319
TreeEffDt 2319
TreeName 2319
TreeSetId 2319
TreeUserKeyValue 2320
LeafCount property 2393
LeafExists method 2368
LeafImageName property 2393
Leaf objects 2302
LeafOnClipboard property 2394
LegendMaxEntries property 459
LegendPosition property 459, 513, 532, 549
LegendStyle property 460, 513, 532
LegendTopSpace property 533
Length property
ColumnNumber class 1990
CompIntfPropInfoCollection objects 651
QueryDBRecordField class 2056
QueryExpression class 2020
Len property 285
Level class
list of properties, returns 2859
methods 2323
properties *See Also* Level class properties
understanding 2323
Level class methods 2323
Level class properties
AllValuesAudit 2324
Description 2324
Name 2324
Number 2324
TreeBranchName 2325
TreeEffDt 2325
TreeName 2325
TreeSetId 2325
TreeUserKeyValue 2325
Level collection
list of methods, properties, returns 2859
methods *See Also* Level collection methods
properties 2322
understanding 2320
Level collection methods
Add 2320
Item 2321
Remove 2321
Level collection properties 2322
LevelComponent property 2412
LevelCount property 2394
LevelMenuBar property 2413
LevelMenuItem property 2413
LevelMenu property 2412
LevelNumber property 2352
Level objects 2323
LevelPage property 2413
Level property
QueryRecordHierarchy class 2031
Rowset class 2167
levels
BranchLevel property 2386
class *See Also* Level class
collection *See Also* Level collection
LevelComponent property 2412
LevelCount property 2394
LevelMenuBar property 2413
LevelMenuItem property 2413
LevelMenu property 2412
LevelNumber property 2352
LevelPage property 2413
Levels property 2394
LevelUse property 2395
objects 2323
ParentLevel property 2398
QueryRecordHierarchy class Level property 2031
Rowset class Level property 2167
Levels property 2394
LevelUse property 2395
licensing
Verity/AllHTTPS property 2515
lights-out mode
showing application messages 1506
understanding 1495
LimitUnapproved property 2041
LineType property 460
Link class
list of properties, returns 2824
properties 1736
understanding 1736
Link class properties 1736
Link collection
list of methods, properties, returns 2824
methods 1734
properties 1735
understanding 1734
Link collection methods
First 1735
Next 1735
Link collection properties 1735
LinkDepth property 2516
link objects 1736
links
class *See Also* Link class
collection *See Also* Links collection
content reference
See Also content reference links
creating 1683
Links property 1691
LinksObjectName property 1736
LinksObjectType property 1736

LinksPortalName property 1737
Links property 1691
LINKTYPE_FIRST property
Utility class 888
LINKTYPE_LAST property
Utility class 888
LINKTYPE_NEXT property
Utility class 888
LINKTYPE_PREVIOUS property
Utility class 889
ListViewAttrs class
list of methods, properties, returns 2878
methods 1431
objects 1431
properties
See Also ListViewAttrs class properties
understanding 1394, 1431
ListViewAttrs class methods 1431
ListViewAttrs class properties
DetailViewLabel 1432
Label 1433
ListViewLabel 1433
Visible 1434
ListViewLabel property 1433
LoadABNChart method
Leaf class 2307
Node class 2337
LoadABNChartOrdered method
Leaf class 2308
Node class 2338
LoadConnectorPropFromNode method 1358
LoadConnectorPropFromRouting method 1359
LoadConnectorPropFromTrx method 1360
LoadConnectorProp method 1357
LoadData method 165
LoadIBContent method 2548
LoadLibrary method 723
load method
Feed class 780
Load method 21
LoadXMLPartString method 1282
LoadXMLString method 1283
Locale property
PDFMerger class 2633
LocalName property 2590
local variables
referencing with PortalRegistry objects 1610
supporting for mobile PeopleCode 1397
using in catch clauses 764
Location property 2496
LockOwner property 2395
LockStatus property 2396
LockTree method 2369
logging
errors for query classes 1901
errors for Search classes 2470
errors for tree classes 2294
PortalRegistry classes errors 1595
subscription PeopleCode errors 2180
Logical property 2011
LogicalQueue class
list of properties, returns 2775
properties 2445
understanding 2445
LogicalQueue constructor 2428
LogicalQueueID property
LogicalQueue class 2446
PhysicalQueue class 2452
LogicalQueue property 2447
logical queues
class *See Also* LogicalQueue class
GetLogicalQueue method 2462
LogicalQueue constructor 2428
MCFFactory class 2446
PhysicalQueue class LogicalQueueID property 2452
Logo property
FeedDoc class 915
LogoutURL property 1070
LongDescription property 1950
AnalyticModelDefn class 91
DataSource class 828
LongName property
ColumnNumber class 1990
DataSourceSetting class 850
QueryDBRecordField class 2056
LongTranslateValue property 962
LookupTableName property 2056
LParenLvl property 2011
LTrim property 2256

M

mail
classes *See Also* mail classes
creating objects 1103
servers 1107
mail classes
constructors 1115
creating an email from rich text editor output, example 1221
creating email and overriding SMTP settings, example 1215
creating email attachments specifying a URL, example 1229
creating email with attachments, example 1227
creating HTML email, example 1216
creating HTML email with images, example 1219
creating mail objects 1103
creating multipart email with text and HTML parts, example 1217
creating multiple email messages, example 1231
creating text email, example 1214
examples 1214
importing 1102
list of constructors, methods, properties, and returns 2725
list of constructors and returns 2726
MCFBodyPart class
See Also MCFBodyPart class
MCFEmail class *See Also* MCFEmail class
MCFGetMail class
See Also MCFGetMail class
MCFInboundEmail
See Also MCFInboundEmail class
MCFMailStore
See Also MCFMailStore class
MCFMailUtil *See Also* MCFMailUtil class
MCFMultiPart *See Also* MCFMultiPart class
MCFOutboundEmail
See Also MCFOutboundEmail class

scope/data types 1102
sending an email with authentication, example 1231
understanding 1101
using 1103
using an SSL connection to send email, example 1232
mail objects, creating 1103
MailServer property 1144
mail servers
deleting email 1140
double-downloading email 1109
retrieving batch email 1114
retrieving email 1107
understanding errors 1114
MainTitleOrient property 461
MainTitle property 461, 514, 533, 550
MainTitleStyle property 461, 514, 533, 550
ManagerInfo property 2205
MapBufRowToUserSortRow method 2150
MapLanguageCd property 2509
MapUserSortRowToBufRow method 2151
Market property 1465
MaxAge property 1083
FeedDoc class 915
FeedEntry class 928
MaxDefinedDecimalLength property
Primitive class 744
MaxDefinedLength property
Primitive class 744
MaxDelta property 91
MaxDropdownDisplayItem property 533
MaxEntries property
FeedDoc class 915
MaxInTreeCriteria property 2041
MaxIterations property 92
MaxJoins property 2041
MaxRowsPerQuery property
CONQRSMGR class 687
MaxRowsToFetch property 2041
MaxSlateDisplayNode property 534
MaxUnapprovedRows property 2042
MCFBodyPart class
constructor 1115
list of methods, properties, and returns 2726
methods
See Also MCFBodyPart Class methods
properties
See Also MCFBodyPart Class properties
properties and returns 2727
understanding 1116
MCFBodyPart class methods
GetHeader 1118
GetHeaderCount 1118
GetHeaderName 1119
GetHeaderNames 1120
GetHeaderValues 1120
GetUnparsedHeaders 1121
MCFBodyPart Class methods
AddHeader 1117
SetAttachmentContent 1122
MCFBodyPart class properties
AttachmentURL 1123
Charset 1124
ContentType 1124
Description 1124
Disposition 1125
Filename 1125
FilePath 1125
FilePathType 1126
IsAttachment 1126
MultiPart 1126
Text 1127
MCFEmail class
properties
See Also MCFEmail class properties
understanding 1127
MCFEmail class properties
BCC 1128
BounceTo 1128
CC 1128
From 1128
Importance 1129
Priority 1129
Recipients 1130
RefIDs 1130
ReplyIDs 1130
ReplyTo 1130
Sender 1131
Sensitivity 1131
Subject 1131
Text 1131
MCFFactory
class
See Also MCFFactory class, MCFFactory class
objects 2421
MCFFactory class
hierarchy 2421
list of properties, returns 2775
properties 2446
understanding 2446
MCFFactory class properties 2446
MCFFactory constructor 2428
MCFFactory objects 2421
MCFGetMail class
constructor 1116
error codes 1114
list of methods, properties, and returns 2727
methods
See Also MCFGetMail class methods
properties
See Also MCFGetMail class properties
understanding 1107
MCFGetMail class methods
CreateQuarantineFolder 1132
errors 1114
GetCount 1132
GetEmailCount 1133
ReadAllEmailHeadersWithAttach 1134
ReadEmails 1135
ReadEmailsWithAttach 1136
ReadEmailsWithUID 1137
ReadEmailWithAttach 1138
ReadHeaders 1139
RemoveEmail 1140
RemoveEmails 1141
SetMCFEmail 1142
MCFGetMail class properties
AttachmentRoot 1143
ContentTypes 1143
ErrorCount 1143
IBNode 1144
MailServer 1144
Password 1145
QuarantineCount 1145

QuarantineFolder 1145
Status 1146
UserID 1146
MCF IM classes
about 1233
lists of functions, methods, properties, and returns 2735
MCFIMSingleButton class constructor 1243
MCFIMInfo
objects 1234
MCFIMInfo class
about 1233
methods 1234
using 1233
MCFIMInfo class methods
AddUser 1235
CheckAll 1235
GetAdditionalUserInfo 1236
GetErrorImageName 1236
GetLaunchURL 1238
GetOffLineImageName 1237
GetOnlineImageName 1237
GetStatus 1239
GetUnknownImageName 1238
RemoveUser 1239
MCFIMInfo objects 1234
MCFIMSingleButton class
about 1234
methods 1240
properties 1244
MCFIMSingleButton class methods
constructor 1243
generateHTML 1240
generateJavaScript 1242
insertXMPPServerUserData 1242
MCFIMSingleButton 1243
updateXMPPServerUserData 1243
MCFIMSingleButton class properties
hideDelay 1244
MCFIMSingleButton method
MCFIMSingleButton class 1243
MCFInboundEmail class
list of methods, properties, and returns 2729
methods
See Also MCFInboundEmail class methods
understanding 1147
MCFInboundEmail class methods
DumpToFile 1147
GetAttachments 1148
GetFrom 1149
GetParts 1149
GetSender 1150
ReadFromDatabase 1151
SaveToDatabase 1151
MCFInboundEmail class properties
AttachList 1152
AttachSizes 1152
DttmReceived 1153
DttmSaved 1153
DttmSent 1153
IBNode 1154
Language 1154
MessageID 1154
NotifyCC 1154
NotifyTo 1155
OffsetReceived 1155
OffsetSent 1155
Server 1156
Size 1156
Status 1156
UID 1156
User 1156
MCF mail classes *See* mail classes
MCFMailStore class
constructor 1116
importing 1157
list of methods and returns 2730
methods
See Also MCFMailStore class methods
properties
See Also MCFMailStore class properties
understanding 1157
MCFMailStore class methods
AuthorizeEmailAttach 1157
DeleteEmail 1158
RetrieveEmail 1159
StoreEmail 1160
MCFMailUtil class
list of methods, properties, and returns 2730
methods
See Also MCFMailUtil class methods
properties
See Also MCFMailUtil class properties
understanding 1161
MCFMailUtil class methods
DecodeText 1161
DecodeWord 1163
EncodeText 1164
EncodeWord 1165
GetErrorMsgParam 1167
IsDomainNameValid 1168
IsEmailServerAvailable 1169
ParseRichTextHtml 1170
ValidateAddress 1171
MCFMailUtil class properties
badaddresses 1173
ErrorDescription 1173
ErrorDetails 1173
ErrorMsgParamsCount 1174
imagesLocation 1174
MessageNumber 1174
MessageSetNumber 1175
MCFMultiPart class
list of methods, properties, and returns 2731
methods
See Also MCFMultiPart class methods
properties
See Also MCFMultiPart class properties
understanding 1175
MCFMultiPart class methods
AddBodyPart 1175
GetBodyPart 1176
GetContentType 1177
GetCount 1177
MCFMultiPart class properties
SubType 1178
MCFOutboundEmail class
constructor 1116
list of methods, properties, and returns 2732
methods
See Also MCFOutboundEmail class methods
properties
See Also MCFOutboundEmail class methods

understanding 1178
MCFOutboundEmail class methods
AddAttachment 1178
AddHeader 1179
GetErrorMsgParam 1181
GetHeader 1181
GetHeaderCount 1182
GetHeaderName 1183
GetHeaderNames 1183
GetHeaderValues 1184
Send 1184
SetSMTPParam 1186
MCFOutboundEmail class properties
BackupSMTPServer 1187
BackupSMTPSSLClientCertAlias 1187
BackupSMTPSSLPort 1187
BackupSMTPUserName 1188
BackupSMTPUserPassword 1188
BackupSMTPUseSSL 1188
BCC 1189
BounceTo 1189
CC 1189
Charset 1189
ContentLanguage 1190
ContentType 1190
DefaultCharSet 1191
Description 1191
Disposition 1191
ErrorDescription 1192
ErrorDetails 1192
ErrorMsgParamsCount 1192
Filename 1193
FilePath 1193
FilePathType 1193
From 1194
Importance 1194
InvalidAddresses 1194
IsAuthenticationReqd 1195
IsOkToSendPartial 1195
IsReturnReceiptReqd 1195
MessageNumber 1196
MessageSetNumber 1196
MultiPart 1196
Priority 1197
Recipients 1197
RefIDs 1198
ReplyIDs 1198
ReplyTo 1198
ResultOfSend 1198
Sender 1199
Sensitivity 1199
SetSMTPParam 1187
SMTPPort 1200
SMTPServer 1200
SMTPSSLClientCertAlias 1200
SMTPSSLPort 1201
SMTPUserName 1201
SMTPUserPassword 1201
SMTPUseSSL 1202
StatusNotifyOptions 1202
StatusNotifyReturn 1203
Subject 1203
Text 1203
TimeToWaitForResult 1203
UsedDefaultConfig 1204
UsedPrimaryServer 1204
ValidSentAddresses 1204
ValidUnsentAddresses 1205
Member property 150
MemberReference class
creating 149
Dimension property 150
Member property 150
method 149
properties 150
members
adding 26
getting 31
removing 35
menus
DetailMenuBar property 2409
DetailMenuItem property 2409
DetailMenu property 2408
GUI/online processing 611
LevelMenuBar property 2413
LevelMenuItem property 2413
LevelMenu property 2412
NodeMenuBar property 2415
NodeMenuItem property 2416
NodeMenu property 2415
SourceMenu property 1458
using %Menu conditions 612
MergePDFs method 2632
MessageBox function 1496
MessageChannel property 1366
Message class
list of methods, properties, and returns 2738
methods *See Also* Message class methods
properties *See Also* Message class properties
message classes
content-based routing 1254
lists of functions, methods, properties, and returns 2737
understanding 1245
Message class methods
Clone method 1258
CopyPartRowset 1259
CopyRowsetDelta method 1261
CopyRowsetDeltaOriginal 1263
CopyRowset method 1260
CreateNextSegment 1266
DeleteSegment 1267
ExecuteEdits method 1268
GenXMLPartString 1271
GenXMLString 1271
GetContentString 1272
GetDocument 1273
GetPartAliasName 1274
GetPartName 1274
GetPartRowset 1275
GetPartVersion 1276
GetPartXMLDoc 1276
GetQueryString 1277
GetRowset 1278
GetSegment 1280
GetXmlDoc 1280
InBoundPublish 1281
LoadXMLPartString 1282
LoadXMLString 1283
Publish method 1283
SetEditTable method 1269, 1286
SetQueryString method 1288
SetStatus method 1289
SetXmlDoc method 1290
SyncRequest method 1291
Update method 1293

UpdateSegment 1294
Message class properties
ActionName 1295
AliasName 1295
ChannelName 1295
Cookies 1296
CurrentSegment 1297
DefaultMessageVersion 1297
DoNotPubToNodeName 1298
GUID 1298
IBException 1299
IBInfo 1299
IsActive 1299
IsBulkLoadTruncation 1300
IsDelta 1300
IsEditError 1301
IsEmpty 1302
IsLocal 1302
IsOperationActive 1303
IsParts 1303
IsPartsStructured 1303
IsRequest 1304
IsSourceNodeExternal 1304
IsStructure 1304
MessageDetail 1305
Name 1305
NRId 1305
OperationName 1305
OperationVersion 1306
ParentTransactionId 1306
PartCount 1306
PubID 1306
PubNodeName 1307
QueueName 1307
QueueSeqIf 1308
ResponseStatus 1308
SegmentCount 1308
SegmentsByDatabase 1308
Size 1309
SubName 1310
SubQueueName 1310
SubscriptionProcessId 1311
TransactionId 1312
Version 1312
MessageDetail property 1305
MessageID property 1154
MessageName property 1366
MessageNumber property
Exception class 769
Field class 962
MCFMailUtil class 1174
MCFOutboundEmail class 1196
PSMessages class 2195
message objects
declaring/scope 1247
populating 1248
Message property
Notification class 1456
ParseResult class 2484
MessageQueue property 1367
messages
copying parts 1259
determining parts 1303
determining structured parts 1303
generating XML part string 1271
getting part alias names 1274
getting part name 1274
getting part rowset 1275
getting part version 1276
getting XML 1276
part count 1306
populating parts 1282
message schema
accessing 1321
verifying existence 1326
message segments
accessing 1280
count 1308
creating 1266
current 1297
deleting 1267
unordering 1371
updating 1294
writing to database 1308
MessageSetNumber property 1196
Exception class 769
Field class 963
MCFMailUtil class 1175
PSMessages class 2195
MessageSeverity property 770
Messages property
AnalyticInstance class 26
AnalyticModel class 36
AnalyticModelDefn class 92
CubeCollection class 52
MessageType property
IBInfo class 1367
PSMessages class 2195
MessageVersion property 1367
messaging
accessing/creating XML messages 2530
content-based routing 1254
CopyToPSFTMessage method 2537
Exception class MessageNumber property 769
Exception class MessageSetNumber property 769
ExternalMessageID property 1364
handlers 1246
handling errors 1256
MessageChannel property 1366
message classes *See Also* message classes
MessageName property 1366
MessageNumber property 962
MessageQueue property 1367
MessageSetNumber property 963
MessageSeverity property 770
MessageType property 1367
MessageVersion property 1367
Notification class Message property 1456
objects *See Also* message objects
OPT_CALL message 1496
ParseResult class Message property 2484
PSMessages *See Also* PSMessages
PSMessagesMode property 2188
PSMessages property 2188
publishing/subscribing to partial records 1251
request messages *See Also* request messages
response messages *See Also* response messages
rowset-based and nonrowset-based messages 1245
sending detailed messages 1501
SOAP 2212
structured and unstructured messages 1245

subscribing to character fields 1251
messaging PeopleCode
deprecated methods and properties 2895
metadata *See Also* RowsetCache class
Query class Metadata property 1950
Query Metadata *See Also* Query Metadata
using query metadata 1898
metadata classes
analytic model 155
Metadata property 1898, 1950
MetaSQL property 1950
Method function 230
MethodInfo class
list of properties, returns 2795
properties 252
MethodInfo collection
list of properties, returns 2794
methods 250
MethodName property 2234
MethodNode property 2234
methods
AESection class
See Also AESection class methods
Agent class 2434
AgentPhysQueueTasks class 2441
AnalyticGrid class 163
AnalyticInstance class 16
AnalyticModel class 26
AnalyticModelDefn class 59
AnalyticTypeDefn class 173
AnalyticTypeRecordDefn class 184
API instantiation 2186
Array class *See Also* Array class methods
AssignedPagelet collection
See Also AssignedPagelet collection methods
assignment class 132
Attribute collection
See Also Attribute collection methods
AvailableCategory collection
See Also AvailableCategory collection methods
AvailablePagelet collection
See Also AvailablePagelet collection methods
Bindings 243
Bindings collection 242
BPELUtil class 292
Business Interlink
See Also Business Interlink methods
Chart class *See Also* Chart class methods
CI collection 1424
ClassInfo collection 247
comparison class 134
CompIntfPropInfoCollection collection
See Also CompIntfPropInfoCollection collection methods
Component Interface class
See Also Component Interface class methods
Component Interface classes
See Also Component Interface classes methods
Component Interface collection
See Also Component Interface collection methods
constant class 137
constructors *See Also* constructors
Content Reference class
See Also Content Reference class methods
Content Reference collection
See Also Content Reference collection methods
ContentReference links 1722
Crypt class 722
cube class 139
CubeCollection class 36
CubeCollectionDefn class 108
CubeDefn class 98
Data collection
See Also Data collection methods, Data collection methods
declaring abstract 202
declaring private 198, 225
declaring protected 194
defining for application classes 199
deprecated 2885, 2895
DynamicCategory collection
See Also DynamicCategory collection methods
Exception class
See Also Exception class methods
Favorite collection
See Also Favorite collection methods
Field class *See Also* Field class methods
File class *See Also* File class methods
Folder class 1663
Folder collection
See Also Folder collection methods
FunctionCall class 147
Gantt class *See Also* Gantt class methods
Grid class 1038
header comments 221
IBConnectorInfo collection
See Also IBConnectorInfo collection methods
IBInfo class *See Also* IBInfo class methods
IBUtil class 298
Incoming Business Interlink class
See Also Incoming Business Interlink class methods
IntBroker class
See Also IntBroker class methods
interfaces 203
Leaf class *See Also* Leaf class methods
Level class 2323
Level collection
See Also Level collection methods
Link collection 1734
ListViewAttrs class 1431
mapping deprecated functions to 2881
MCFGetMail class
See Also MCFGetMail class methods
MCFIMInfo class 1234
MCFIMSingleButton class 1240
MCFMailStore
See Also MCFMailStore class methods
MCFMailUtil
See Also MCFMailUtil class methods
MCFMultiPart
See Also MCFMultiPart class methods
MCFOutboundEmail
See Also MCFOutboundEmail class methods
MemberReference class 149
Message class

See Also Message class methods
Method function 230
MethodInfo collection 250
Mobile class *See Also* Mobile class methods
Namespaces class 246
Namespaces collection 244
naming 208
Node class *See Also* Node class methods
Node collection
See Also Node collection methods
NodeTemplate collection
See Also NodeTemplate collection methods
Notification class 1453
NotificationTemplate class
See Also NotificationTemplate class methods
Operation class 151
OptBase callback 1543
OptBase class
See Also OptBase class methods
OptEngine class
See Also OptEngine class methods
OptInterface class
See Also OptInterface class methods
OrganizerDefn class 125
OrgChart class
See Also OrgChart class methods
PageletCategory class 1767
PageletCategory collection
See Also PageletCategory collection methods
Pagelet class 1777
Pagelet collection
See Also Pagelet collection methods
ParseResult collection 2482
PeerDefaultAttributes class 1439
PermissionValue collection
See Also PermissionValue collection methods
PhysicalQueue class 2447
Portal class 1654
Portal collection
See Also Portal collection methods
PortalRegistry class
See Also PortalRegistry class methods
PortalRegistry collection
See Also PortalRegistry collection methods
PostReport class
See Also PostReport class methods
PrcsApi class
See Also PrcsApi class methods
ProcessRequest class
See Also ProcessRequest class methods
PropertyAttrs class 1435
PropertyInfo collection 253
PropertyInfoCollection class 1417
PSMessages collection
See Also PSMessages collection methods
Query class *See Also* Query class methods
Query collection
See Also Query collection methods
QueryCriteria class
See Also QueryCriteria class methods
QueryCriteria collection
See Also QueryCriteria collection methods
QueryDBRecord class
See Also QueryDBRecord class methods
QueryDBRecord collection
See Also QueryDBRecord collection methods
QueryDBRecordField class
See Also QueryDBRecordField class methods
QueryDBRecordField collection
See Also QueryDBRecordField collection methods
QueryExpression collection
See Also QueryExpression collection methods
QueryField class
See Also QueryField class methods
QueryField collection
See Also QueryField collection methods
QueryList class
See Also QueryList class methods
Query Metadata collection
See Also Query Metadata collection methods
QueryPrompt collection
See Also QueryPrompt collection methods
QueryRecord class 1977
QueryRecord collection
See Also QueryRecord collection methods
QueryRecordHierarchy collection
See Also QueryRecordHierarchy collection methods
QuerySelect class
See Also QuerySelect methods
QuerySelect collection
See Also QuerySelect collection methods
RatingBoxChart class
See Also RatingBoxChart class methods
Record class *See Also* Record class methods
RemoteNode collection
See Also RemoteNode collection methods
Request class
See Also Request class methods
Response class methods
See Also Response class methods
RolePermissionValue collection
See Also RolePermissionValue collection methods
Row class *See Also* Row class methods
Rowset class *See Also* Rowset class methods
RuleDefn class 129
SearchField collection
See Also SearchField collection methods
SearchIndex class 2494
SearchIndexes collection
See Also SearchIndexes collection methods
Search Language collection
See Also Search Language collection methods
SearchQuery class
See Also SearchQuery class methods
SearchRecordField collection
See Also SearchRecordField collection methods
SearchResult collection 2485
Search Schedule collection
See Also Search Schedule collection methods
Search Start Options collection
See Also Search Start Options collection

methods
SelectedPagelet collection
See Also SelectedPagelet collection methods
self-referencing 210
sending optimization status 1501
Session class
See Also Session class methods, 1398
SMTPSession
See Also SMTPSession class methods
SOAPDoc class
See Also SOAPDoc class methods
SOAPDoc object Method section 2216
SQL class *See Also* SQL class methods
super-referencing 218
SyncServer class
See Also SyncServer class methods
TabDefinition class 1737
TabDefinition collection
See Also TabDefinition collection methods
Task class *See Also* Task class methods
TaskList class 2459
Tree class *See Also* Tree class methods
Tree Structure class
See Also Tree Structure class methods
understanding 213
understanding API Repository 234
UserHomepage class 1789
UserTab collection
See Also UserTab collection methods
using for collections
See Also PeopleCode collections
Util class 2462
variable class 153
Worklist class 1467
WorklistEntry class
See Also WorklistEntry class methods
Methods property 249
MIMEList property
Search FS Options class 2519
Search HTTP Options class 2516
MIMEListType property
Search FS Options class 2520
Search HTTP Options class 2517
MIMETYPE_ATOM property
Utility class 889
MIMETYPE_OPML property
Utility class 889
MIMETYPE_XML property
Utility class 890
mobile
classes *See Also* mobile classes
device 1386
events *See Also* mobile events
Mobile Agent *See Also* Mobile Agent
objects *See Also* mobile objects
pages *See Also* mobile pages
Mobile Agent
detail views 1387
list views 1387
synchronization 2261, 2264
synchronization considerations 2283
understanding 1385
mobile applications
system ID field name 2108
timestamp field name 2109
Mobile class
list of deprecated functions, methods, properties, returns, variables 2874
methods *See Also* Mobile class methods
properties 1416
understanding 1394, 1399
mobile classes
hierarchy 1390
list of deprecated functions, methods, properties, returns, variables 2874
ListViewAttrs *See Also* ListViewAttrs class
Mobile *See Also* Mobile class
PeerDefaultAttributes
See Also PeerDefaultAttributes class
PropertyAttrs *See Also* PropertyAttrs class
PropertyInfo *See Also* PropertyInfo class
PropertyInfoCollection
See Also PropertyInfoCollection class
understanding 1385
Mobile class methods
Create 1399
Find 1400
Get 1401
GetName 1402
GetParent 1403
GetPeerDefaultAttributesByName 1404
GetPropertyAttrsByName 1404
GetPropertyByName 1405
GetPropertyInfoByName 1406
GetTopParent 1407
IsNew 1407
IsNewAndNeverSaved 1408
IsSameAs 1409
ReadBlobFromFile 1409
Save 1410
SetModified 1412
SetPropertyByName 1412
ValidateEnum 1413
ValuesDifferFromLastSync 1414
WasSaved 1415
WriteBlobToFile 1415
Mobile collections
CI collection *See Also* CI collection
Data collection *See Also* Data collection
mobile device 1386
mobile events
OnConflict 2265
OnGetDefinition 2285
OnGetProperty 2269
OnGetPropertyFilter 2270
OnSelect 2267
OnSetProperty 2270
OnValidate 2268
OnValidateSet 2284
understanding 1389, 2265
using attachment events 2269
mobile objects
declaring/scope 1397
hierarchy 1390
understanding 1389
Mobile objects
understanding 1399
mobile pages
understanding 1385
views 1387
mobile PeopleCode
debugging 1394, 1395
functions 1443
handling errors 1396
PeopleCode classes 1442

Session class methods 1398
supported system variables 1442
supporting variables 1397
tracing 1396
models
analytic type definitions, deleting 176
MoreRowsAvailable property 1951
MouseOverMsgNum property 963
MouseOverMsgSet property 964
MoveAsChildByName method
Leaf class 2310
Node class 2340
MoveAsChild method
Leaf class 2309
Node class 2339
MoveAsSibByName method 2341
MoveAsSibByRange method 2312
MoveAsSib method
Leaf class 2311
Node class 2341
MoveFirst method 352
MoveNext method 353
MoveToDoc method 354
MSFFactory
example 2463
MsgSchemaExists method 1326
MsgSet property
CONQRS_CONST class 700
MultiChannel Framework functions
Broadcast 2444
MultiChannel Framework mail classes
See mail classes
MultiPart property
MCFBodyPart class 1126
MCFOutboundEmail class 1196
Multipurpose Internet Mail Extensions (MIME) 1108

N

Name class 1091
Name property
Agent class 2438
AnalyticModelDefn class 93
AnalyticTypeDefn class 183
AnalyticTypeModelDefn class 183
AnalyticTypeRecordDefn class 186
AttributeValue class 1705
Bindings 243
ClassInfo 249
Collection class 760
ColumnNumber class 1990
CompIntfPropInfoCollection objects 652
Compound class 752
CONQRSMGR class 687
Content Reference class 1691
ContentReference links 1728
Cookie class 1084
cube class 141
CubeCollectionDefn class 122
CubeDefn class 107
DataSourceParameter class 838
DataSourceParameterValue class 844
DataSourceSetting class 850
DimensionDefn class 94
ExplicitDimensionSet class 97
Field class 964
File class 1012
Folder class 1671
FunctionCall class 149
GridColumn class 1047
Level class 2324
Message class 1305
MethodInfo 252
Namespaces 246
Node class 1648, 2352
NodeTemplate class 1659
OrganizerDefn class 129
Page class 1584
PageletCategory class 1771
Pagelet class 1782
PermissionValue class 1712
Portal class 1655
PortalRegistry class 1641
Primitive class 745
PropertyInfo 255
PropertyInfo object 1422
Query class 1951
QueryCriteria class 2011
QueryDBRecord class 2047
QueryDBRecordField class 2056
QueryExpression class 2020
Query Metadata class 2035
QueryPrompt class 2065
QueryRecord class 1979
QueryRecordHierarchy class 2031
Record class 2106
Rowset class 2167
SearchField class 2491
SearchIndex class 2496
TabDefinition class 1743
Tree class 2396
Tree Structure class 2414
UserFunctionDefn class 125
variable class 154
NameSpaceID property
Feed class 796
namespaces
API Repository 234
class *See Also* Namespaces class
collection *See Also* Namespaces collection
defining XML namespaces 2531
Namespaces property 241
Namespaces class
list of properties, returns 2794
methods 246
properties 246
Namespaces class properties
Classes 246
Name 246
Namespaces collection
list of properties, returns 2793
methods 244
properties 244
Namespaces collection methods
Item 244
ItemByName 245
Namespaces collection properties 244
Namespaces methods 246
Namespaces property 241
NamespaceURI property 2591
naming
content references 1608
Java packages and classes 1086

NDMaxDisplayDescLength property 550
Negation property 2012
NetworkServices property 2205
NewWindowLink property 1585
Next method
- Array class 271
- AssignedPagelet collection 1754
- Attribute collection 1710
- AvailableCategory collection 1757
- AvailablePagelet collection 1762
- CompIntfPropInfoCollection collection 648
- Component Interface collection 614
- Content Reference collection 1703
- DynamicCategory collection 1766
- Favorite collection 1807
- Folder collection 1681
- Link collection 1735
- Node collection 1651
- NodeTemplate collection 1662
- PageletCategory collection 1776
- Pagelet collection 1788
- ParseResult collection 2482
- PermissionValue collection 1716
- Portal collection 1657
- PortalRegistry collection 1644
- PSMessages collection 2194
- Query collection 1921
- QueryCriteria collection 1997
- QueryDBRecord collection 2044
- QueryDBRecordField collection 2050
- QueryExpression collection 2017
- QueryField collection 1983
- QueryList class 2025
- Query Metadata collection 2034
- QueryPrompt collection 2060
- QueryRecord collection 1975
- QueryRecordHierarchy collection 2029
- QuerySelect collection 1959
- RemoteNode collection 1653
- RolePermissionValue collection 1720
- SearchField collection 2490
- SearchIndexes collection 2493
- Search Language collection 2508
- SearchRecordField collection 2503
- SearchResult collection 2486
- Search Schedule collection 2512
- Search Start Options collection 2523
- SelectedPagelet collection 1801
- TabDefinition collection 1748
- UserTab collection 1795

Next property 2322
NextSibling property 2591
NextSib property
- Leaf class 2317
- Node class 2353

NextStep method 724
NextUrl property
- FeedDoc class 916

NickName property 2438
Node class
- list of methods, properties, returns 2860
- list of properties, returns 2812
- methods *See Also* Node class methods
- properties
  - *See Also* Node class properties, Node class properties
- understanding 1645, 2326

Node class methods
- Branch 2326
- Cut 2327
- Delete 2327
- DeleteByName 2328
- Expand 2328
- GenABNMenuElement 2329
- GenBreadCrumbs 2330
- GenRelatedActions 2331
- InsertChildLeaf 2331
- InsertChildNode 2332
- InsertChildRecord 2333
- InsertDynChildLeaf 2334
- InsertSib 2335
- InsertSibRecord 2336
- LoadABNChart 2337
- LoadABNChartOrdered 2338
- MoveAsChild 2339
- MoveAsChildByName 2340
- MoveAsSib 2341
- MoveAsSibByName 2341
- PasteChild 2342
- PasteSib 2343
- RefreshDescription 2344
- Rename 2344
- SwitchLevel 2345
- Unbranch 2345

Node class properties
- ActiveNode 1646
- AllChildCount 2346
- AllChildNodeCount 2346
- AppsRelease 1646
- ChildLeafCount 2347
- ChildNodeCount 2347
- ColImageName 2347
- ContentURI 1646
- DefaultPortalName 1646
- Description 1647, 2347
- DisplayLevelNumber 2347
- ExpImageName 2348
- FirstChildLeaf 2348
- FirstChildNode 2348
- HasChildLeaves 2349
- HasChildNodes 2349
- HasChildren 2349
- HasNextSib 2349
- HasPrevSib 2350
- IsBranched 2350
- IsChanged 2350
- IsCut 2350
- IsDefault 1647
- IsDeleted 2350
- IsInserted 2351
- IsLocal 1647
- IsRoot 2351
- LastChildLeaf 2351
- LastChildNode 2352
- LevelNumber 2352
- Name 1648, 2352
- NextSib 2353
- NodePassword 1648
- NodeType 1648
- Parent 2353
- PortalURI 1649
- PrevSib 2353
- State 2354
- ToolsRelease 1649
- TreeBranchName 2354
- TreeEffDt 2354

TreeName 2355
TreeSetId 2355
TreeUserKeyValue 2355
Type 2355
NodeColImageName property 2397
Node collection
list of methods, properties, returns 2813
methods *See Also* Node collection methods
properties 1651
understanding 1649
Node collection methods
First 1649
ItemByName 1650
Next 1651
NodeComponent property 2414
NodeCount property 2397
Node descriptor style property 534, 535, 536
NodeDescrnStyle property 534, 535, 536
NodeDN property 1368
NodeExists method 2375
NodeExpImageName property 2397
NodeField property 2415
NodeMenuBar property 2415
NodeMenuItem property 2416
NodeMenu property 2415
NodeMultiNavigate property 2416
NodeName property
Business Interlink class 370
XmlNode class 2591
node objects 2326
NodeOnClipboard property 2397
NodePage property 2416
NodePassword property 1648
NodePath property 2591
NodeRecord property 2417
nodes
BodyNode property 2231
class *See Also* Node class
collection *See Also* Node collection
DestinationNode property 1364
EnvelopeNode property 2231
FinalDestinationNode property 1365
GetActualRemoteNodes method 1612
GetLocalNode method 1613
GetNodes method 1613
GetRemoteNodes method 1615
HeaderNode property 2233
HostNodeName property 1655
MethodNode property 2234
naming content references 1608
NodeColImageName property 2397
NodeComponent property 2414
NodeCount property 2397
NodeDN property 1368
NodeExpImageName property 2397
NodeField property 2415
NodeMenuBar property 2415
NodeMenuItem property 2416
NodeMenu property 2415
NodeMultiNavigate property 2416
NodeName property 370
NodeOnClipboard property 2397
NodePage property 2416
NodeRecord property 2417
NodeType property 370
NodeUserKeyField property 2417
NodeValue property 371
objects 2326
OrigNode property 1369
Portal collection *See Also* Portal collection
remote 1612
RemoteNode collection
See Also RemoteNode collection
RequestingNodeDescription property 1370
RequestingNodeName property 1370
SourceNode property 1371
templates *See Also* NodeTemplate
understanding 1589
using the URL property 1607
VisitedNodes property 1372
NodeTemplate
class *See Also* NodeTemplate class
collection *See Also* NodeTemplate collection
NodeTemplates property 1641
NodeTemplate class
list of properties, returns 2814
properties 1658
understanding 1658
NodeTemplate class properties 1658
NodeTemplate collection
list of methods, properties, returns 2814
methods
See Also NodeTemplate collection methods
properties 1662
understanding 1659
NodeTemplate collection methods
DeleteItem 1659
First 1660
InsertItem 1661
ItemByName 1661
Next 1662
NodeTemplate collection properties 1662
node templates *See* NodeTemplate
NodeTemplates property 1641
NodeType property
Incoming Business Interlink class 370
Node class 1648
XmlNode class 2592
NodeUserKeyField property 2417
NodeValue property
Incoming Business Interlink class 371
XmlNode class 2592
NonRepudiationID property 1368
nonrowset-based messages
understanding 1245
NonSSBs property 2205
notification
PeopleCode 1256
NotificationAddress class
list of properties, returns 2749
properties 1459
understanding 1459
NotificationAddress class properties 1459
NotificationAddress constructor 1448
Notification class
importing 1453
list of methods, properties, returns 2747
methods 1453
properties
See Also Notification class properties
understanding 1453
Notification class constructors
Notification 1447
NotificationAddress 1448
NotificationTemplate 1450

Worklist 1451
WorklistEntry 1451
WSWorklistEntry 1452
notification classes
list of constructors, methods, properties, returns 2747
Notification classes
constructors
See Also Notification class constructors
creating notification objects 1447
examples 1484
importing 1446
Notification *See Also* Notification class
NotificationAddress
See Also NotificationAddress class
NotificationTemplate
See Also NotificationTemplate class
scope 1445
understanding 1445
Worklist *See Also* Worklist class
WorklistEntry *See Also* WorklistEntry class
Notification class properties
ContentType property 1454
dttmCreated property 1454
EmailReplyTo property 1455
FileNames property 1455
FileTitles property 1455
language_cd property 1456
Message property 1456
NotifyBCC property 1456
NotifyCC property 1456
NotifyFrom property 1457
NotifyGuid property 1457
NotifyTo property 1457
Rte property 1458
SourceComponent property 1458
SourceMarket property 1458
SourceMenu property 1458
Subject property 1458
Template property 1459
Notification constructor 1447
notification objects, creating 1447
notification PeopleCode 1256
notifications
classes *See Also* Notification classes
declaring 1446
NotificationTemplate class
list of methods, properties, returns 2749
methods
See Also NotificationTemplate class methods
properties
See Also NotificationTemplate class properties
understanding 1460
NotificationTemplate class methods
GetAndExpandTemplate 1461
SetupCompVarsAndRcpts 1462
SetupGenericVars 1463
NotificationTemplate class properties
ComponentId 1464
Instruction 1464
Language 1465
Market 1465
Priority 1465
Responses 1465
Subject 1465
TemplateId 1466
TemplateType 1466
Text 1466
NotificationTemplate constructor 1450
NotifyBCC property 1456
NotifyCC property 1154, 1456
NotifyFrom property 1457
NotifyGuid property 1457
Notify method 2275
NotifyTextMsgNum property 1868
NotifyTextMsgSet property 1869
NotifyTo property 1155, 1457
notifyToWindow method 1883
NRId property 1305
NumberOfAttachments property 1368
Number property 2324

O

object cleanup
Java programs 1097
ObjectRowset property
CONQRSMGR class 688
objects, Java *See* Java objects
objects, PeopleCode *See* PeopleCode objects
ObjectType property
DataSource class 829
DataSourceParameter class 838
DataSourceParameterValue class 844
DataSourceSetting class 850
Feed class 796
FeedDoc class 916
FeedEntry class 929
OffsetReceived property 1155
OffsetSent property 1155
OLLineType property 462
OLType property 462
OnConflict event 2265
onDelete method
DataSource class 824
On Error method
WSWorklistEntry class 1482
OnGetDefinition event 2285
OnGetProperty event 2269
OnGetPropertyFilter event 2270
OnNotify method
WSWorklistEntry class 1483
OnRequestSend method
AsyncFFSend class 291
onSave method
DataSource class 825
OnSelect event 2267
OnSetProperty event 2270
OnValidate event 2268
OnValidateSet event 2284
OpenAsOfDate method
Tree class 2371
OpenForExport method
Tree class 2373
Open method
AESection class 7
CONQRSMGR class 676
Crypt class 725
File class 989
PortalRegistry class 1634
Query class 1933
SQL class 2252

Tree class 2369
Tree Structure class 2405
OpenWholeTree method 2374
Operand1 property
comparison class 135
Operation class 152
Operand2 property
comparison class 135
Operation class 152
OPERATINGMODE_AUTHORIZATION property
Utility class 890
OPERATINGMODE_DEFAULT property
Utility class 890
OPERATINGMODE_DELETION property
Utility class 890
OPERATINGMODE_EXECUTION_NOENTRY property
Utility class 891
OPERATINGMODE_EXECUTION property
Utility class 891
OperatingMode property
Feed class 796
Operation class
creating 150
method 151
Operand1 property 152
Operand2 property 152
properties 152
Type property 152
OperationName property 1305
OperationType property 1368
OperationVersion property 1306, 1369
Operator property
description 2012
using 1895
OPIs
developing PeopleCode to use 1543
invoking 1493
oprid property 1479
Oprid property 1460
OprID property
CONQRSMGR class 688
OPRID property
SCHED_INFO class 707
OPT_CALL message
sending messages 1507, 1508
showing messages for lights-out mode 1506
understanding lights-out mode 1496
OPT_CALL program 1506
OptBase callback methods 1543
OptBase class methods
callback 1543
GetParmDate 1544
GetParmDateArray 1545
GetParmDateTime 1546
GetParmDateTimeArray 1546
GetParmInt 1548
GetParmIntArray 1549
GetParmNumber 1547
GetParmNumberArray 1548
GetParmString 1550
GetParmStringArray 1550
GetParmTime 1551
GetParmTimeArray 1552
Init 1552
OptDeleteCallback 1553
OptInsertCallback 1554
OptPostUpdateCallback 1554
OptPreUpdateCallback 1555
OptRefreshCallback 1556
SetOutputParmDate 1557
SetOutputParmDateArray 1557
SetOutputParmDateTime 1558
SetOutputParmDateTimeArray 1559
SetOutputParmInt 1561
SetOutputParmIntArray 1561
SetOutputParmNumber 1559
SetOutputParmNumberArray 1560
SetOutputParmString 1562
SetOutputParmStringArray 1563
SetOutputParmTime 1563
SetOutputParmTimeArray 1564
understanding 1543
OptDeleteCallback method 1544, 1553
OptEngine class
methods *See Also* OptEngine class methods
properties
See Also OptEngine class properties
OptEngine class methods
CheckOptEngineStatus 1518
FillRowset 1520
GetDate 1522
GetDateArray 1523
GetDateTime 1524
GetDateTimeArray 1525
GetNumber 1526
GetNumberArray 1527
GetString 1528
GetStringArray 1529
GetTime 1531
GetTimeArray 1532
GetTraceLevel 1533
RunAsynch *See Also* RunAsynch method
RunSynch *See Also* RunSynch method
SetTraceLevel 1537
ShutDown 1539
using optimization PeopleCode in Application Engine programs 1490
using optimization PeopleCode on application servers 1489
OptEngine class properties
DetailedStatus 1542
DetailMsgs 1541
OptEngine objects 1492
optimization
PeopleCode
See Also optimization PeopleCode
optimization engines
creating 1498
loading 1502
loading analytic instances 1491
OptEngine class *See Also* OptEngine class
running transactions in asynchronous mode 1534
running transactions in synchronous mode 1536
shutting down 1494, 1500, 1539
status, checking 1518
status, viewing 1498
status messages, logging 1567
using the CreateOptEngine function 1508
using the GetOptEngine function 1513
optimization models
activating 1565
active 1573

changing variable types 1576
deactivating 1567
overriding bounding values 1574
restoring bounding values 1573
retrieving solution details 1571
retrieving solutions 1569
solving 1577
optimization PeopleCode
analytic instances, creating 1491
analytic instances, deleting 1494
analytic instances, loading 1491
invoking OPIs 1493
OptBase application class
See Also OptBase class methods
programming for database updates 1495
request messages *See Also* request messages
response messages
See Also response messages
running optimization transactions 1492
shutting down optimization engines 1494
understanding the functions 1508
using in Application Engine 1492
using in Application Engine programs 1490
using lights-out mode 1495
using on application servers 1489, 1491
optimization PeopleCode classes
list of functions, methods, properties, returns 2752
optimization transactions
processing parameters 1507
running 1492, 1498, 1504
running in asynchronous mode 1534
running in synchronous mode 1536
running on application servers 1493
running on the Application Engine 1493
OptimizeHorizontalSpace property 537
OptimizeVerticalSpace property 537
OptInsertCallback method 1544, 1554
OptInterface class methods
ActivateModel 1565
ActivateObjective 1566
DeactivateModel 1567
DumpMsgToLog 1567
FindRowNum 1568
GetSolution 1569
GetSolutionDetail 1571
IsModelActive 1573
RestoreBounds 1573
SetVariableBounds 1574
SetVariableType 1576
Solve 1577
understanding 1565
OPTIONS table 973
OptPostUpdateCallback method 1544, 1554
OptPreUpdateCallback method 1544, 1555
OptRefreshCallback method 1544, 1556
Oracle
sorting records 2136
using hints in expressions 2022
OrderByDirection property 1990
OrderByNumber property 1991
OrderNumber property
DataSourceParameterValue class 844
organization chart
creating 411
events 416
example 588
methods 519
properties 525
subrecord definitions 417
Organization Chart
understanding 407
organization chart events 416
organizer
accessing 80
accessing names 81
adding to analytic model 62
deleting 72
renaming 87
OrganizerDefn class
AttachPart method 125
Comments property 129
DetachPart method 126
GetPartNames method 127
methods 125
Name property 129
parts
attaching 125
detaching 126
getting names 127
using 128
properties 129
UsesPart method 128
OrgChart class
list of methods 2677
list of properties 2677
methods *See Also* OrgChart class methods
objects 425
properties *See Also* OrgChart class properties
OrgChart class methods 521
SetCrumbData 519
SetCrumbRecord 520
SetDropdownData 520
SetDropdownRecord 521
SetLegendImg 522
SetNodeData 523
SetNodeRecord 524
SetPopUpNodeData 524
SetPopUpNodeRecord 525
OrgChart class properties
CenterFocusNode 526
ChartScrollType 526
Collapsed_Msg 527
CollapsedImage 527
CollapseMainIconSpace 527
CrumbDescrSelectStyle 542
CrumbDescrStyle 528
CrumbMaxDisplayLength 528
CrumbSeparatorImage 528
Direction 529
DropDownBoxStyle 529
Expanded_Msg 530
ExpandedImage 530
HasLegend 530
Height 530
ImageHeight 531, 532
ImageLocation 531
LegendPosition 532
LegendStyle 532
LegendTopSpace 533
MainTitle 533
MainTitleStyle 533
MaxDropdownDisplayItem 533
MaxSlateDisplayNode_Msg 534
Node descriptor style 534, 535, 536
NodeDescrnStyle 534, 535, 536

OptimizeHorizontalSpace 537
OptimizeVerticalSpace 537
Pop-up Node Descriptor Style 538, 539, 540
PopupHeaderStyle 538
PopupNodeDescrnStyle 538, 539, 540
Style 541
SuppressPeopleCodeOnNode 541
Width 529, 542
OriginalTime property 2458
OriginalValue property 965
originatorid property 1479
OrigNode property 1369
OrigProcess property 1369
OrigTimeStamp property 1369
OrigUser property 1370
OrigValue property
Primitive class 745
outbound email 1104
OutDestFormat property
PostReport class 1829
ProcessRequest class 1871
OutDestination property 2611
OutDestinationType property 2612
OutDest property 1869
OutDestType property 1871
OUTDESTTYPE property
SCHED_INFO class 707
outer joins query 2074
OutProcessFileName property
CONQRSMGR class 688
OutputField property 2021
Output method 766
OutputUnicode property 1951
OutputUNICODE property 2205
overflow
email errors 1114
OverflowedTaskList property 2452
OverFlowTime property 2459
OverflowedTaskList property 2452
OverFlowTime property 2459
OwnerId property
Content Reference class 1691
ContentReference links 1728
Folder class 1671
PageletCategory class 1771
Pagelet class 1782
PortalRegistry class 1641
TabDefinition class 1743
OwnerID property
AnalyticTypeDefn class 183
Feed class 797

P

PackageName property
DocumentKey class 740
packages
application *See Also* application packages
Java packages delivered with PeopleTools 1086
Page class
list of functions, methods, properties, returns 2756
properties *See Also* Page class properties
shortcut considerations 1581
understanding 1581
Page class properties
CopyURLLink 1582
CustomizePageLink 1583
DisplayOnly 1583
HelpLink 1584
Name 1584
NewWindowLink 1585
Visible 1585
PageIndex property 2641
PageletCategories property 1642
PageletCategory
class *See Also* PageletCategory class
collection
See Also PageletCategory collection
PageletCategory class
list of methods, properties, returns 2830
methods 1767
properties
See Also PageletCategory class properties
understanding 1767
PageletCategory class properties
Attributes 1768
Author 1769
AuthorAccess 1769
Authorized 1769
CascadedPermissions 1769
CreationDate 1770
Description 1770
Label 1770
Name 1771
OwnerId 1771
Pagelets 1771
Permissions 1772
Product 1772
PublicAccess 1772
SequenceNumber 1773
PageletCategory collection 1773
list of methods, properties, returns 2831
methods
See Also PageletCategory collection methods
properties 1776
understanding 1773
PageletCategory collection methods
DeleteItem 1773
First 1774
InsertItem 1775
ItemByName 1775
Next 1776
Pagelet class
list of methods, properties, returns 2832
methods 1777
properties *See Also* Pagelet class properties
understanding 1777
Pagelet class properties
Attributes 1778
Author 1778
AuthorAccess 1779
Authorized 1779
CascadedPermissions 1779
ContentProvider 1779
CreationDate 1780
DefaultColumn 1780
Description 1780
EditPageContentProvider 1780
EditPageQueryString 1781
HelpID 1781
IsHideMinimize 1781

Label 1781
Name 1782
OwnerId 1782
ParentName 1782
Permissions 1782
Product 1783
PublicAccess 1783
QualifiedURL 1783
SequenceNumber 1783
URL 1784
URLType 1784
Pagelet collection
list of methods, properties, returns 2833
methods *See Also* Pagelet collection methods
properties 1788
understanding 1785
Pagelet collection methods
DeleteItem 1785
First 1786
InsertItem 1786
ItemByName 1787
Next 1788
PageletLabel property 1760
PageletName property
AssignedPagelet class 1750
AvailablePagelet class 1760
SelectedPagelet class 1798
pagelet objects
AssignedPagelet objects 1749
AvailablePagelet 1758
PageletCategory 1767
understanding 1777
pagelets *See Also* pagelet objects
AvailablePagelet collection
See Also AvailablePagelet collection
class *See Also* Pagelet class
collection *See Also* Pagelet collection
PageletCategory class
See Also PageletCategory class
PageletCategory collection
See Also PageletCategory collection
PageletName property 1750
SelectedPagelet class PageletName property 1798
SelectedPagelet collection
See Also SelectedPagelet collection
SelectedPagelet objects 1796
using FindPgltByName method 1629
Pagelets property 1771
PageNumber class
about 2627
import statements 2627
PageNumber class methods
constructor 2627
PageNumber 2627
PageNumber class properties
BackgroundFile 2628
FontName 2628
FontSize 2629
PositionX 2630
PositionY 2630
StartFromPageNum 2630
StartNum 2631
PageNumber method
PageNumber class 2627
PageNumber property 2634
page objects
declaring/scope 1582
understanding 1581
pages
Page class *See Also* Page class
page objects *See Also* page objects
parameters
passing in application class methods 200
passing with object data types 200
using the out specification 201
ParametersCompleted property
DataSource class 829
Parameters property
DataSource class 829
ParentFlag property 2031
ParentLevel property 2398
ParentName property
Content Reference class 1692
ContentReference links 1729
Folder class 1671
Pagelet class 1782
Tree class 2398
ParentNode property 2593
Parent property
DataSource class 829
DataSourceParameter class 838
DataSourceParameterValue class 845
DataSourceSetting class 850
Leaf class 2318
Node class 2353
ParentRecord property 965
ParentRow property
Record class 2107
Rowset class 2168
ParentRowset property
Row class 2122
Rowset class 2168
ParentSelectNum property 1971
ParentTransactionId property 1306
ParmCount property 2234
Parse method 2476
ParseResult
class *See Also* ParseResult class
collection *See Also* ParseResult collection
objects 2484
ParseResult class
list of properties, returns 2801
properties 2484
understanding 2484
ParseResult class properties 2484
ParseResult collection
list of methods, properties, returns 2801
methods 2482
properties 2483
understanding 2482
ParseResult collection methods 2482
ParseResult collection properties 2483
ParseResult objects 2484
ParseRichTextHtml method 1170
ParseXmlFromFile method
Document class 735
ParseXmlFromURL method 2549
Document class 735
ParseXmlString method 2551
Document class 736
ParseXMLString method 367
PartCount property 1306
parts
attaching 125
detaching 126

getting names 127
using 128
Password property 1145
PasteChild method 2342
PasteSib method
Leaf class 2312
Node class 2343
PathInfo property
IConnectorInfo collection 1384
Request class 1070
Path property
Content Reference class 1692
ContentReference links 1729
Cookie class 1084
Folder class 1671
paths, relative 982
PCExtFcnCalls property 2206
PCFcnReturnValues property 2206
PCFetchedValues property 2206
PCIntFcnCalls property 2206
PCListProgram property 2207
PCParameterValues property 2207
PCProgramStatements property 2207
PCStack property 2207
PCStartOfPrograms property 2208
PCTraceProgram property 2208
PCVariableAssignments property 2208
PDFFont property 1952
PDFMerger class
about 2631
import statements 2631
PDFMerger class methods
constructor 2631
PDFMerger 2631
PDFMerger method
MergePDFs 2632
PDFMerger class 2631
PDFMerger properties
ConfProp 2633
Locale 2633
PageNumber 2634
Watermark 2634
PeerDefaultAttributes class
list of deprecated methods, properties, returns 2879
methods 1439
properties 1440
understanding 1394, 1439
PeopleCode
about cxvii
accessing PeopleCode classes using Java *See Also* PeopleCode objects
accessing the API Repository 235
API, about cxvii
calling via Java 1090
developer's guide, about cxvii
functions *See Also* functions
generating for file layout automatically 980
language reference, about cxvii
mapping Java types/classes 1094
mobile *See Also* mobile PeopleCode
notification 1256
optimization *See Also* optimization PeopleCode
quick reference for syntax 2645
typographic conventions for this guide cxvii
PeopleCode classes
accessing via Java 1091
AESection *See Also* AESection class
Agent class *See Also* Agent class
AgentPhysQueueProps class *See Also* AgentPhysQueueProps class
AgentPhysQueueTasks class *See Also* AgentPhysQueueTasks class
API Repository classes *See Also* API Repository classes
application classes *See Also* application classes
Array *See Also* Array class, Array class
AssignedPagelet *See Also* AssignedPagelet class
AttributeValue *See Also* AttributeValue class
AvailableCategory *See Also* AvailableCategory class
AvailablePagelet *See Also* AvailablePagelet class
Bindings *See Also* Bindings class
BPEL classes *See Also* BPEL classes
Broadcast class *See Also* Broadcast class
Business Interlink *See Also* Business Interlink class
charting *See Also* charting classes
ClassInfo *See Also* ClassInfo class
Component Interface *See Also* Component Interface class
Component Interface classes *See Also* Component Interface classes
connected query *See Also* connected query classes
Content Reference *See Also* Content Reference class
ContentReference links *See Also* ContentReference class
Cookie *See Also* Cookie class
Crypt *See Also* Crypt class
Data Item *See Also* Data Item
deprecated 2871
document *See Also* document classes
Exception *See Also* Exception class, Exception class
Favorite *See Also* Favorite class
feed *See Also* feed classes
Field *See Also* Field class
File *See Also* File class
Folder *See Also* Folder class
Func Java 1091
Grid *See Also* Grid class
GridColumn *See Also* GridColumn class
header comments 222
IBInfo *See Also* IBInfo class
Incoming Business Interlink *See Also* Incoming Business Interlink class
IntBroker *See Also* IntBroker class
iScript classes *See Also* iScript classes
Java *See Also* Java class
Leaf *See Also* Leaf class
Level *See Also* Level class
Link *See Also* Link class
LogicalQueue class *See Also* LogicalQueue class
mail classes *See Also* mail classes
MCFFactory *See Also* MCFFactory class
MCFFactory class *See Also* MCFFactory class
MCF IM classes *See Also* MCF IM classes
message classes *See Also* message classes

MethodInfo *See Also* MethodInfo class
mobile classes *See Also* mobile classes
Name Java 1091
Namespaces *See Also* Namespaces class
Node *See Also* Node class, Node class
NodeTemplate *See Also* NodeTemplate class
Notification classes
See Also Notification classes
Page *See Also* Page class
Pagelet *See Also* Pagelet class
PageletCategory
See Also PageletCategory class
ParseResult *See Also* ParseResult class
PermissionValue
See Also PermissionValue objects, PermissionValue class
PhysicalQueue class
See Also PhysicalQueue class
Portal *See Also* Portal class
PortalRegistry
See Also PortalRegistry classes, 1591, *See Also* PortalRegistry class
PostReport *See Also* PostReport class
PrcsApi class 1880
process request
See Also process request classes
PropertyInfo *See Also* PropertyInfo class
PSMessages *See Also* PSMessages class
query classes *See Also* query classes
QueryCriteria *See Also* QueryCriteria class
QueryDBRecord
See Also QueryDBRecord class
QueryDBRecordField
See Also QueryDBRecordField class
QueryExpression
See Also QueryExpression class
QueryList *See Also* QueryList class
QueryListValue
See Also QueryListValue class
Query Metadata
See Also Query Metadata class
QueryPrompt *See Also* QueryPrompt class
QueryRecord *See Also* QueryRecord class
QueryRecordHierarchy
See Also QueryRecordHierarchy class
QuerySecurityProfile
See Also QuerySecurityProfile class
QuerySelect *See Also* QuerySelect class
QueryStatistics
See Also QueryStatistics class
quick reference 2645
Record *See Also* Record class, Record class
Regional Settings
See Also Regional Settings class
Repository *See Also* Repository class
Request *See Also* Request class
Response *See Also* Response class
RolePermissionValue
See Also RolePermissionValue objects
Row *See Also* Row class
Rowset *See Also* Rowset class
RowsetCache class
See Also RowsetCache class
SearchField *See Also* SearchField class
Search FS Options
See Also Search FS Options class
Search HTTP Options
See Also Search HTTP Options class
SearchIndex *See Also* SearchIndex class
Search Language
See Also Search Language class
SearchQuery *See Also* SearchQuery class
SearchRecordField
See Also SearchRecordField class
SearchRecord Options
See Also SearchRecord Options class
SearchResult *See Also* SearchResult class
Search Schedule
See Also Search Schedule class
Search Start Options 2524
SelectedPagelet
See Also SelectedPagelet class
Session *See Also* Session class, Session class
SOAPDoc *See Also* SOAPDoc class
SQL *See Also* SQL class
SyncServer *See Also* SyncServer class
SysCon Java 1091
SysVar Java 1091
TabDefinition *See Also* TabDefinition class
Task class *See Also* Task class
TaskList class *See Also* TaskList class
Trace Setting
See Also Trace Setting class properties
TransformData
See Also TransformData class
Tree *See Also* Tree class
Tree Structure *See Also* Tree Structure class
universal queue classes
See Also universal queue classes
UserHomepage
See Also UserHomepage class
UserTab *See Also* UserTab class
using security with PortalRegistry 1591
Util class *See Also* Util class
Verity search classes
See Also Verity search classes
XmlDoc classes *See Also* XmlDoc class
XML Publisher classes
See Also XML Publisher classes

PeopleCode collections
accessing PortalRegistry objects 1592
API Repository collections
See Also API Repository collections
AssignedPagelet
See Also AssignedPagelet collection
Attribute *See Also* Attribute collection
AvailableCategory
See Also AvailableCategory collection
AvailablePagelet
See Also AvailablePagelet collection
Bindings *See Also* Bindings collection
Bindings collection
See Also Bindings collection
Branch *See Also* Branch collection
ClassInfo *See Also* ClassInfo collection
CompIntfPropInfoCollection
See Also CompIntfPropInfoCollection collection
Component Interface
See Also Component Interface collection
Content Reference
See Also Content Reference collection
Data
See Also Data collection, Data collection, Data collection
declaring 2184

DynamicCategory
See Also DynamicCategory collection
Favorite *See Also* Favorite collection
Folder *See Also* Folder collection
Level *See Also* Level collection
Links *See Also* Links collection
MethodInfo *See Also* MethodInfo collection
Namespaces *See Also* Namespaces collection
nectorInfo
See Also IBConnectorInfo collection
Node *See Also* Node collection
NodeTemplate
See Also NodeTemplate collection
Pagelet *See Also* Pagelet collection
PageletCategory
See Also PageletCategory collection
ParseResult *See Also* ParseResult collection
PermissionValue
1592, *See Also* PermissionValue collection
Portal *See Also* Portal collection
PortalRegistry
See Also PortalRegistry collection
PortalRegistry collections
See Also PortalRegistry collections
PropertyInfo
See Also PropertyInfo collection
PSMessages *See Also* PSMessages collection
query collections *See Also* query collections
QueryCriteria
See Also QueryCriteria collection
QueryDBRecord
See Also QueryDBRecord collection
QueryDBRecordField
See Also QueryDBRecordField collection
QueryExpression
See Also QueryExpression collection
QueryField *See Also* QueryField collection
Query Metadata
See Also Query Metadata collection
QueryPrompt
See Also QueryPrompt collection
QueryRecord
See Also QueryRecord collection
QueryRecordHierarchy
See Also QueryRecordHierarchy collection
QuerySelect *See Also* QuerySelect collection
RemoteNode
See Also RemoteNode collection
RolePermissionValue
1592, *See Also* RolePermissionValue collection
Search collections
See Also Search collections
SearchField collection
See Also SearchField collection
SearchIndexes
See Also SearchIndexes collection
Search Language
See Also Search Language collection
SearchRecordField
See Also SearchRecordField collection
SearchResult
See Also SearchResult collection, SearchResult collection
Search Schedule
See Also Search Schedule collection
Search Start Options
See Also Search Start Options collection
SelectedPagelet
See Also SelectedPagelet collection
TabDefinition
See Also TabDefinition collection
Tree collections *See Also* Tree collections
UserTab *See Also* UserTab collection
using methods/properties 216
PeopleCode events
GUI/online processing 611
mobile *See Also* mobile events
processing Search dialog boxes 611
using the WorklistEntry class Save method 1471
using the WorklistEntry class SaveWithCustomData method 1473
PeopleCode functions
accessing messages 1397
application class
See Also application class functions
binding meta-SQL 2242
calling from Java programs 1096
Create 204
CreateXmlDoc 2531
declaring external 206
deprecated 2871, 2881
DistributeDataByRules 2267
DoSave 1444
function libraries 1053
Generate 1057
GUI/online processing 611
Import 228
instantiating record objects 2077
instantiating rowset objects 2127
iScripts *See Also* iScripts
Java *See Also* Java functions
mapping deprecated/invalid terms to new terms 2869
mapping old names to new names 2869
Method 230
methods *See Also* methods
mobile PeopleCode 1443
no longer supported 2898
ScheduleProcess 1836
structuring via application classes 190
using flattening and promotion 264
using transfer functions 763
PeopleCode interfaces 202
PeopleCode methods
deprecated 2871
PeopleCode objects
accessing 1592
AESection *See Also* AESection objects
ApiObject 2183
arrays *See Also* array class
AssignedPagelet 1749
Attribute collection 1707
AttributeValue 1704
AvailableCategory 1755
AvailablePagelet 1758
BPEL 288
Business Interlink
See Also Business Interlink objects
chart 397, 406, 425
collections *See Also* PeopleCode collections
CompIntfPropInfoCollection 644
Component Interface
See Also Component Interface objects
content reference

See Also content reference objects
content reference links 1721
creating mail objects 1103
Data collection 1425
declaring cookie objects 1060
declaring iScript objects 1060
declaring Request objects 1060
declaring Response objects 1060
favorite 1802
field 935
folder 1663
Folder collection 1678
grid *See Also* grid objects
IBInfo 1336
instantiating API objects 2186
IntBroker 1312
leaf 2302
Level 2323
links 1736
ListViewAttrs class 1431
MCFFactory 2421
mobile *See Also* mobile objects
nnectorInfo colle 1374
node 2326
nodes 1645
NodeTemplate 1658
notification 1447
page *See Also* page objects
PageletCategory 1767
PageletCategory collection 1773
pagelets *See Also* pagelet objects
ParseResult 2484
PermissionValue
See Also PermissionValue class
portal 1590
PostReport 1826
PrcsApi 1881
ProcessRequest 1836
PropertyAttrs class 1434
PropertyInfo class 1419
PropertyInfoCollection 1417
PSMessages 2180
queries *See Also* query objects
QueryCriteria
1893, *See Also* QueryCriteria objects
QueryDBRecord 2045
QueryDBRecordField 2052
QueryExpression 2018
QueryField 1984
query metadata 2035
QueryRecord 1976
QueryRecordHierarchy 2027
QuerySelect 1960
RatingBoxChart 430
records *See Also* record objects
regional settings 2182
request *See Also* request objects
response *See Also* response objects
rows *See Also* row objects
rowsets *See Also* rowset objects
search 2471
SearchField 2491
Search FS Options 2518
Search HTTP Options 2514
SearchIndex 2465
search indexes 2465
Search Language 2509
search queries 2475
SearchRecordField 2504
search record options 2497
search schedule 2513
Search Start 2524
SelectedPagelet 1796
Session *See Also* session objects
SOAPDoc *See Also* SOAPDoc objects
SQL *See Also* SQL objects
SyncServer *See Also* SyncServer objects
TabDefinition 1737
TemplateObject property 1696
TemplateObject property (ContentReference links) *See Also* TemplateObject property
TraceSettings 2182
trees *See Also* tree objects
tree structure 2402
universal queue 2424
UserHomepage 1789
UserTab 1791
using with Java 1091
XMLDoc *See Also* XMLDoc objects
PeopleCode programs
cleaning up in Java 1097
PeopleCode properties
deprecated 2871
PeopleSoft API Respository *See* API Repository
PeopleSoft Mobile Agent *See Also* Mobile Agent
PeopleSoft MultiChannel Framework mail classes
See mail classes
PeopleSoft Query Manager 1885
PeopleSoft Tree Manager 2291
performance issues
accessing a Component Interface structure 644
BulkExecute method 319
BulkMode property 2239
IsPlainImage property 458, 512
PerformanceMethod property 2398
PerformanceSelectorOption property 2399
PerformanceSelector property 2399
replicating business rules 223
RequestedFields property 2480
reusing a cursor 2244
RowsetCache class 2171
setting trace levels 1537
using closed trees 2292
using Get/Set keywords/methods 215
using OnValidate event 2268
using OnValidateSet event 2284
using synchronization server events 2264
Visible property 1048
PerformanceMethod property 2398
PerformanceSelectorOption property 2399
PerformanceSelector property 2399
PermissionListDelete method 1635
permission lists
PermissionListDelete method 1635
PermissionListSaveAs method 1635
using PermissionValue 1592
PermissionListSaveAs method 1635
permissions
CascadedPermissions folder property 1667
CascadedPermissions PortalRegistry property 1593
CascadedPermissions property 1687, 1725, 1769, 1779
CascadedRolePermissions folder property 1668

CascadedRolePermissions property 1688, 1726
GrantPermissionForComponent method 1631
GrantPermissionForScript method 1633
Permissions folder class property 1672
Permissions PortalRegistry property 1593
Permissions property 1692, 1729, 1772, 1782
PermissionValue *See Also* PermissionValue
RevokePermissionForComponent method 1636
RevokePermissionForScript method 1637
RolePermissions Folder class property 1673
RolePermissions property 1694, 1731
setting via PermissionValue objects 1817
Permissions property
Content Reference class 1692
ContentReference links 1729
Folder class 1672
PageletCategory class 1772
Pagelet class 1782
understanding 1593
PermissionValue
class *See Also* PermissionValue class
collection
See Also PermissionValue collection
objects *See Also* PermissionValue objects
using 1592
PermissionValue class
list of, properties, returns 2821
properties
See Also PermissionValue class properties
understanding 1711
PermissionValue class properties
Cascade 1711
Name 1712
PermType 1712
PermissionValue collection
list of methods, properties, returns 2822
methods
See Also PermissionValue collection methods
properties 1716
understanding 1712
PermissionValue collection methods
DeleteItem 1713
First 1714
InsertItem 1714
ItemByName 1715
Next 1716
PermissionValue objects
setting permissions 1817
PermType property 1712
persist aggregate
getting 114
setting 119
PhysicalQueue class
list of methods, properties, returns 2776
methods 2447
properties
See Also PhysicalQueue class properties
understanding 2447
PhysicalQueue class methods 2447
PhysicalQueue class properties
AcceptedTaskList 2449
Agent 2449
AssignedTaskList 2449
BrowserURL 2450
EnqueuedTaskList 2450
EscalatedTaskList 2450
InternalURL 2451
IsActive 2451
LogicalQueueID 2452
OverflowedTaskList 2452
PhysicalQueueID 2452
RENURLID 2452
TotalAgents 2453
PhysicalQueue constructor 2429
PhysicalQueueID property 2439
AgentPhysQueueTasks class 2443
LogicalQueue class 2446
PhysicalQueue class 2452
Task class 2459
TaskList class 2460
PhysicalQueueID property
PhysicalQueueID class 2440
physical queues
constructors 2429
PhysicalQueue class
See Also PhysicalQueue class
PhysicalQueueID property
2439, 2440, 2443, 2446, 2459, 2460
TotalPhysicalQueues property 2439
understanding 2446
viewing properties 2439
viewing tasks 2441
PixelsPerRow property 514
pluggable cryptography 721
Pop-up Node Descriptor Style property 538, 539, 540
Pop method 272
populatePrefData method
Feed class 781
PopupHeaderStyle property 538
PopUpHeaderStyle property 551
PopUpHeight property 551, 555
PopupNodeDescrnStyle property 538, 539, 540
PopUpStyle property 552
PopUpWidth property 552
Portal class
list of methods, properties, returns 2813
methods 1654
properties 1655
understanding 1654
Portal class methods 1654
Portal class properties 1655
Portal collection
list of methods, properties, returns 2814
methods *See Also* Portal collection methods
properties 1658
understanding 1656
Portal collection methods
First 1656
ItemByName 1656
Next 1657
Portal collection properties 1658
portal component content references 1605
PortalFolder property 1872
portal objects 1590
PortalRegistry
classes
See Also PortalRegistry classes, PortalRegistry classes, PortalRegistry classes
collection *See Also* PortalRegistry collection
portal registry
classes *See Also* PortalRegistry classes

FindPortalRegistries method 1611
GetPortalRegistry method 1614
PortalRegistry classes
See Also PortalRegistry
understanding 1587
using security for objects 1591
PortalRegistry class
changing properties 1808
list of methods, properties, returns 2809
methods
See Also PortalRegistry class methods
properties
See Also PortalRegistry class properties
understanding 1615
PortalRegistry classes
AssignedPagelet
See Also AssignedPagelet class
AttributeValue *See Also* AttributeValue class
AvailableCategory
See Also AvailableCategory class
AvailablePagelet
See Also AvailablePagelet class
Content Reference
See Also Content Reference class
ContentReference links
See Also ContentReference class
examples 1808
Favorite *See Also* Favorite class
Folder *See Also* Folder class
handling errors 1595
Link *See Also* Link class
list of methods, properties, returns 2809
Node *See Also* Node class
NodeTemplate *See Also* NodeTemplate class
Pagelet *See Also* Pagelet class
PageletCategory
See Also PageletCategory class
PermissionValue
See Also PermissionValue class
Portal class *See Also* Portal class
PortalRegistry *See Also* PortalRegistry class
saving content 1609
SelectedPagelet
See Also SelectedPagelet class
Session class *See Also* Session class
TabDefinition *See Also* TabDefinition class
understanding 1587, 1591
UserHomepage
See Also UserHomepage class
UserTab *See Also* UserTab class
using 1591
using ValidFrom/ValidTo 1595
viewing the class hierarchy 1598
PortalRegistry class methods
BuildSearchIndex 2474
Close 1616
CopyObject 1617
Create 1618
CreateContentRefLink 1619
CreateRemote 1620
Delete 1621
DeleteHomepage 1622
FindCRefByName 1623
FindCRefByURL 1624
FindCRefForURL 1625
FindCRefLinkByName 1627
FindFolderByName 1627
FindPgltByName 1629
GetAbsoluteContentURL 1630
GetDefaultHPTabOID 1630
GetQualifiedURL 1631
GetSearchQuery 2474
GrantPermissionForComponent 1631
GrantPermissionForScript 1633
Open 1634
PermissionListDelete 1635
PermissionListSaveAs 1635
RevokePermissionForComponent 1636
RevokePermissionForScript 1637
Save 1638
PortalRegistry class properties
DefaultTemplate 1639
Description 1640
Favorites 1640
FolderNavObject 1640
Homepage 1640
IsFolderNavigation 1641
Name 1641
NodeTemplates 1641
OwnerId 1641
PageletCategories 1642
Portals 1642
RootFolder 1642
TabDefinitions 1642
TemplateObject 1643
PortalRegistry collection
list of methods, properties, returns 2811
methods
See Also PortalRegistry collection methods
properties 1645
understanding 1643
PortalRegistry collection methods
First 1643
Item 1644
Next 1644
PortalRegistry collections
AssignedPagelet
See Also AssignedPagelet collection
Attribute *See Also* Attribute collection
AvailableCategory
See Also AvailableCategory collection
AvailablePagelet
See Also AvailablePagelet collection
Content Reference
See Also Content Reference collection
DynamicCategory
See Also DynamicCategory collection
Favorite *See Also* Favorite collection
Folder *See Also* Folder collection
Links *See Also* Links collection
Node *See Also* Node collection
NodeTemplate collection
See Also NodeTemplate collection
Pagelet *See Also* Pagelet collection
PageletCategory
See Also PageletCategory collection
PermissionValue
See Also PermissionValue collection
Portal collection *See Also* Portal collection
PortalRegistry
See Also PortalRegistry collection
RemoteNode
See Also RemoteNode collection
RolePermissionValue
See Also RolePermissionValue collection
SelectedPagelet

See Also SelectedPagelet collection
TabDefinition
See Also TabDefinition collection
UserTab *See Also* UserTab collection
PortalRegistry objects
declaring 1610
deleting 1609
life cycle 1596
referencing via C/C++ 1820
referencing via Visual Basic 1820
scope 1610
portals
AbsolutePortalURL property 1685, 1724
class *See Also* Portal class
DefaultPortalName property 1646
LinksPortalName property 1737
objects 1590
portal component 1605
PortalFolder property 1872
PortalURI property 1649
registry *See Also* portal registry
templates 1604
PortalSpecificPersonalization property
DataSource class 830
Portals property 1642
PortalURI property 1649
Port property 1211
PositionX property 2630
PositionY property 2630
Post Office Protocol 3 (POP3) 1108
PostReport
class *See Also* PostReport class
objects 1826
PostReport class
determining distribution lists 1825
examples 1832
list of functions, methods, properties, returns 2757
methods *See Also* PostReport class methods
properties
See Also PostReport class properties
understanding 1825
PostReport class methods
AddDistributionOption 1827
Put 1827
PostReport class properties
ExpirationDate 1828
OutDestFormat 1829
ProcessInstance 1829
ProcessName 1830
ProcessType 1830
ReportDescr 1830
ReportFolder 1831
ReportId 1831
ServerName 1832
SourceReportPath 1832
PostReport objects 1826
PrcsApi
class *See Also* PrcsApi class
objects 1881
PrcsApi class
constructors 1882
importing 1881
list of constructors, methods, returns 2761
methods *See Also* PrcsApi class methods
understanding 1880
PrcsApi class constructors 1882
PrcsApi class methods
getAllFileNames 1882
notifyToWindow 1883
PrcsApi constructor 1882
PrcsApi objects 1881
PRCSFILENAME property
SCHED_INFO class 707
Prefix property 2593
PreviousSibling property 2593
PreviousUrl property
FeedDoc class 916
prevoprid property 1479
PrevSib property
Leaf class 2318
Node class 2353
Primitive class
about 741
list of methods 2689
list of properties 2689
Primitive class methods
GetEnumName 741
GetParent 742
GetPath 742
Primitive class properties
ElementType 743
EnumCount 743
IsChanged 743
IsInitialized 744
IsRequired 744
MaxDefinedDecimalLength 744
MaxDefinedLength 744
Name 745
OrigValue 745
PrimitiveSubType 745
PrimitiveType 746
SequenceNumber 747
Value 747
PrimitiveSubType property
Primitive class 745
PrimitiveType property
Primitive class 746
PrintJobHTMLRpt method 1848
PrintJobRqstRpt method 1849
PrintOutput method 2604
PrintSchdlHTMLRpt method 1852
Priority property 1465
MCFEmail class 1129
MCFOutboundEmail class 1197
private
declaring instance variables
See Also instance variables, instance variables
declaring methods 198, 225
PROCESS_INSTANCE property
SCHED_INFO class 708
ProcessedCount property 2480
ProcessInstanceID property 2617
ProcessInstance property 2612
PostReport class 1829
ProcessRequest class 1872
process instances 1829
ProcessName property
PostReport class 1830
ProcessRequest class 1872
ProcessReport method 2605
ProcessRequest
objects 1836
process request
classes *See Also* process request classes

ProcessRequest class
examples 1877
format values 1841
methods
See Also ProcessRequest class methods
output type values 1841
properties
See Also ProcessRequest class properties
scheduling a single process 1878
scheduling jobs with item changes 1879
scheduling jobs without item changes 1878
process request classes
list of functions, methods, properties, returns 2758
understanding 1835
ProcessRequest class methods
AddDistributionOption 1846
AddNotifyInfo 1847
PrintJobHTMLRpt 1848
PrintJobRqstRpt 1849
PrintSchdlHTMLRpt 1852
RunJobSetNow 1855
Schedule 1855
SetEmailOption 1857
SetItemFolder 1858
SetNotifyAppMethod 1859
SetNotifyService 1862
SetOutputOption 1863
UpdateRunStatus 1865
ProcessRequest class properties
EmailAttachLog 1865
EmailSubject 1866
EmailText 1866
EmailWebReport 1867
FileName 1867
JobName 1868
LanguageCd 1868
NotifyTextMsgNum 1868
NotifyTextMsgSet 1869
OutDest 1869
OutDestFormat 1871
OutDestType 1871
PortalFolder 1872
ProcessInstance 1872
ProcessName 1872
ProcessType 1873
RunControlID 1874
RunDateTime 1874
RunLocation 1875
RunRecurrence 1875
RunStatus 1875
Status 1877
TimeZone 1877
ProcessRequest objects 1836
ProcessType property
PostReport class 1830
ProcessRequest class 1873
Product property
Content Reference class 1693
ContentReference links 1730
Folder class 1672
PageletCategory class 1772
Pagelet class 1783
TabDefinition class 1743
programs, Application Engine
See Application Engine programs
Promotion 264
Prompt property 652
PromptRecordFieldName property 2065
PromptRecord property 1953
Prompts property 1953
PromptTableName property 966
PromptTable property 2065
DataSourceParameter class 839
DataSourceSetting class 851
PROPDEFAULT property
CONQRSPROPERTY class 698
properties 152
AESection class
See Also AESection class properties
Agent class *See Also* Agent class properties
AgentPhysQueueProps class 2440
AgentPhysQueueTasks class 2442
analytic grid class 167
AnalyticInstance class 25
AnalyticModel class 35
AnalyticModelDefn class 90
analytic type definition 182
AnalyticTypeModelDefn 183
Array class 284
AssignedPagelet class
See Also AssignedPagelet class properties
AssignedPagelet collection 1754
assignment class 133
Attribute collection 1710
AttributeValue class
See Also AttributeValue class properties
AvailableCategory class 1755
AvailableCategory collection 1758
AvailablePagelet class
See Also AvailablePagelet class properties
AvailablePagelet collection 1762
Bindings 243
Bindings collection 242
Business Interlink class 372
CascadedPermissions property 1593
Cascade property 1593
Chart class *See Also* Chart class properties
CI collection 1424
ClassInfo 249
ClassInfo collection 247
comparison class 135
CompIntfPropInfoCollection collection 648
Component Interface class
See Also Component Interface class properties
Component Interface classes
See Also Component Interface class properties
Component Interface collection 615
constant class 138
Content Reference class
See Also Content Reference class properties
Content Reference collection 1704
ContentReference links 1723
content references
See Also content reference properties
Cookie class
See Also Cookie class properties
Crypt class 727
cube class 141
CubeCollectionDefn class 121
cube collections 51
CubeDefn class 106
Data collection 642, 1430

declaring abstract 202
declaring protected 194
deprecated 2885, 2895
DimensionDefn class 93
DynamicCategory collection 1767
Exception class
See Also Exception class properties
Favorite class
See Also Favorite class properties
Favorite collection 1808
Field class *See Also* Field class properties
File class *See Also* File class properties
Folder class *See Also* Folder class properties
Folder collection 1682
FunctionCall class 149
Grid class 1044
GridColumn class
See Also GridColumn class properties
IBConnectorInfo collection 1382
IBInfo class *See Also* IBInfo class properties
Incoming Business Interlink class
See Also Incoming Business Interlink properties, Incoming Business Interlink class properties
Leaf class *See Also* Leaf class properties
Level class *See Also* Level class properties
Level collection 2322
Link class 1736
Link collection 1735
ListViewAttrs class
See Also ListViewAttrs class properties
LogicalQueue class 2445
mapping deprecated functions to 2881
MCFFactory class 2446
MCFIMSingleButton class 1244
MCFMailStore
See Also MCFMailStore class properties
MCFMailUtil
See Also MCFMailUtil class properties
MCFMultiPart
See Also MCFMultiPart class properties
MCFOutboundEmail
See Also MCFOutboundEmail class properties
MemberReference class 150
Message class
See Also Message class properties
MethodInfo class 252
MethodInfo collection 250
Mobile class 1416
Namespaces class 246
Namespaces collection 244
naming 209
Node class
See Also Node class properties, Node class properties
Node collection 1651
NodeTemplate class 1658
NodeTemplate collection 1662
NotificationAddress class 1459
Notification class
See Also Notification class properties
NotificationTemplate class
See Also NotificationTemplate class properties
OrganizerDefn class 129
overriding 214, 218
Page class *See Also* Page class properties
PageletCategory class
See Also PageletCategory class properties
PageletCategory collection 1776
Pagelet class
See Also Pagelet class properties
Pagelet collection 1788
ParseResult class 2484
ParseResult collection 2483
PeerDefaultAttributes class 1440
Permissions property 1593
PermissionValue class
See Also PermissionValue class properties
PermissionValue collection 1716
PhysicalQueue class
See Also PhysicalQueue class properties
Portal class 1655
Portal collection 1658
PortalRegistry class
See Also PortalRegistry class properties
PortalRegistry collection 1645
PostReport class
See Also PostReport class properties
ProcessRequest
See Also ProcessRequest class properties
PropertyAttrs class
See Also PropertyAttrs class properties
PropertyInfo class 255
PropertyInfo collection 253
PropertyInfoCollection class 1418
PropertyInfo objects
See Also PropertyInfo objects properties
PSMessages class
See Also PSMessages class properties
PSMessages collection 2194
Query class *See Also* Query class properties
Query collection 1921
QueryCriteria class
See Also QueryCriteria class properties
QueryCriteria collection 1997
QueryDBRecord class 2047
QueryDBRecord collection 2045
QueryDBRecordField class
See Also QueryDBRecordField class properties
QueryDBRecordField collection 2051
QueryExpression class
See Also QueryExpression class properties
QueryExpression collection 2018
QueryField class
See Also QueryField class properties
QueryField collection 1983
QueryList class 2025
QueryListValue class 2026
Query Metadata class
See Also Query Metadata class properties
Query Metadata collection 2034
QueryPrompt class
See Also QueryPrompt class properties
QueryPrompt collection 2061
QueryRecord class
See Also QueryRecord class properties
QueryRecord collection 1976
QueryRecordHierarchy class
See Also QueryRecordHierarchy class properties
QueryRecordHierarchy collection 2030
QuerySecurityProfile class
See Also QuerySecurityProfile class

properties
QuerySelect class
See Also QuerySelect class properties
QuerySelect collection 1960
QueryStatistics class
See Also QueryStatistics class properties
Record class
See Also Record class properties
Regional Settings class
See Also Regional Settings class properties
RemoteNode collection 1654
Repository class 241
Request class
See Also Request class properties
Response class properties 1082
RolePermissionValue collection 1721
Row class *See Also* Row class properties
Rowset class
See Also Rowset class properties
RuleDefn class 131
SearchField class 2491
SearchField collection 2490
Search FS Options class
See Also Search FS Options class properties
Search HTTP Options class
See Also Search HTTP Options class properties
SearchIndex class
See Also SearchIndex class properties
SearchIndexes collection 2493
Search Language class 2509
Search Language collection 2508
SearchQuery class
See Also SearchQuery class properties
SearchRecordField class
See Also SearchRecordField class properties
SearchRecordField collection 2503
SearchRecord Options class
See Also SearchRecord Options class properties
SearchResult class
See Also SearchResult class properties
SearchResult collection 2486
Search Schedule class
See Also Search Schedule class properties
Search Schedule collection 2512
Search Start Options class 2524
Search Start Options collection 2523
SelectedPagelet class
See Also SelectedPagelet class properties
SelectedPagelet collection 1801
Session class
See Also Session class properties
SOAPDoc class
See Also SOAPDoc class properties
SQL class *See Also* SQL class properties
SyncServer class
See Also SyncServer class properties
TabDefinition class
See Also TabDefinition class properties
TabDefinition collection 1748
Task class *See Also* Task class properties
TaskList class
See Also TaskList class properties
TransformData class
See Also TransformData class properties
Tree class *See Also* Tree class properties
Tree Structure class
See Also Tree Structure class properties
understanding 213
understanding API Repository 234
UserFunctionDefn class 124
UserHomepage class 1790
UserTab class
See Also UserTab class properties
UserTab collection 1796
using for collections
See Also PeopleCode collections
using Get/Set 215
using PortalRegistry object security properties 1591
using read-only 215
variable class 154
WorklistEntry class
See Also WorklistEntry class properties
properties, OptEngine class
See OptEngine class properties
Properties class
about 2635
import statements 2635
Properties class methods
constructor 2635
GetProperty 2636
Properties 2635
SetProperty 2637
Properties method
Properties class 2635
Properties property 249
property
Branch collection 2302
Gantt class *See Also* Gantt class properties
OrgChart class
See Also OrgChart class properties
RatingBoxChart class
See Also RatingBoxChart class properties
PropertyArray property
CONQRSMGR class 689
PropertyAttrs class
list of methods, properties, returns 2878
methods 1435
objects 1434
properties
See Also PropertyAttrs class properties
PropertyAttrs class objects 1434
PropertyAttrs class properties
DisplayOnly 1436
InError 1437
Label 1437
ShowRequiredCue 1438
Visible 1438
PropertyCount property
Compound class 752
PropertyInfo class
list of properties, returns 2795, 2876
objects 1419
properties 255
understanding 1394, 1419
PropertyInfo class objects 1419
PropertyInfo collection
list of methods, properties, returns 2876
list of properties, returns 2795
methods 253
properties 253
PropertyInfoCollection class

methods 1417
understanding 1394, 1417
PropertyInfoCollection class properties 1418
PropertyInfoCollection objects 1417
PropertyInfoCollection property
CompIntfPropInfoCollection objects 652
Component Interface class 635
Mobile class property 1417
PropertyInfo object 1423
PropertyInfo object properties
DefaultValue property 1419
HasDefaultValue property 1420
IsCollection property 1420
IsPeerReference property 1421
IsRefToParent property 1421
IsSimpleApplicationProperty property 1421
IsSystemProperty property 1422
Name property 1422
PropertyInfoCollection property 1423
PropertyValue property 2280
PROPNAME property
CONQRSPROPERTY class 698
PROPVALUEARRAYDESC property
CONQRSPROPERTY class 699
PROPVALUEARRAY property
CONQRSPROPERTY class 699
PROPVALUE property
CONQRSPROPERTY class 698
protected
declaring methods 194
declaring properties 194
protected statements 764
Protocol property 1070
ProxyHost property 2517
ProxyPort property 2517
PS_CLASSPATH environment variable 1086
PS_MESSAGE_LOG table 1496
psappsrv.cfg
Add To CLASSPATH parameter 1086
PSMessages
class *See Also* PSMessages class
collection *See Also* PSMessages collection
objects 2180
PSMessagesMode property 2188
PSMessages property 2188
PSMessages class
list of properties, returns 2790
properties
See Also PSMessages class properties
PSMessages class properties
ExplainText 2194
MessageNumber 2195
MessageSetNumber 2195
MessageType 2195
Source 2196
Text 2197
PSMessages collection
accessing 2191
displaying error messages 2181
list of methods, properties, returns 2789
logging errors for PortalRegistry classes 1595
logging errors for query classes 1901
logging errors for Search classes 2470
logging errors for Session class 2180
logging errors for tree classes 2294
methods
See Also PSMessages collection methods
properties 2194
PSMessages collection methods
DeleteAll 2191
DeleteItem 2192
First 2192
Item 2193
Next 2194
PSMessagesMode property 2188
PSMessages objects 2180
PSMessages property 2188
PT_DETMSGS record 1501
PT_MCF_MAIL application package 1101
PT_OPT_BASE:OptBase application class
See OptBase class methods
PT_OPTCALL program 1502
PT_OPTDETMSGS record 1496
PT_OPTPARMS record
checking optimization engine status 1498
getting the trace level 1500
running transactions 1498
sending optimization status 1501
setting the trace level 1499
shutting down optimization engines 1500
understanding OPT_CALL messages 1496
understanding request messages 1498
understanding response messages 1501
PubID property 1306
public
declaring external interfaces 193
properties, creating 213
PublicAccess property
Content Reference class 1693
ContentReference links 1730
Folder class 1672
PageletCategory class 1772
Pagelet class 1783
setting object access 1591
TabDefinition class 1743
publication
publishing to partial records 1251
Publish method 1326
PublicationID property 1370
PublicPrivate property 1953
PublishedInSites property
Feed class 797
Published property
FeedEntry class 929
Publish method
IntBroker class 1326
Message class 1283
ReportDefn class 2606
publishToSites method
Feed class 782
PubNodeName property 1307
Push method 273
Put method 1827

Q

QualifiedURL property
Content Reference class 1693
Favorite class 1803
Pagelet class 1783
TabDefinition class 1744
UserTab class 1792
QuarantineCount property 1145

QuarantineFolder property 1145
queries
  adding an operator and Expression 2
    dependencies 1895
  adding criteria 2068
  adding expressions 1895
  collections *See Also* query collections
  creating 2066
  database queries 1892
  having criteria 1903
  life cycle 1887
  objects *See Also* query objects
  outer joins 2074
  Query class *See Also* Query class
  query classes *See Also* query classes
  QueryCriteria class
    See Also QueryCriteria class
  QueryCriteria collection
    See Also QueryCriteria collection
  QueryDBRecord class
    See Also QueryDBRecord class
  QueryDBRecord collection
    See Also QueryDBRecord collection
  QueryDBRecordField class
    See Also QueryDBRecordField class
  QueryDBRecordField collection
    See Also QueryDBRecordField collection
  QueryExpression class
    See Also QueryExpression class
  QueryExpression collection
    See Also QueryExpression collection
  QueryField class *See Also* QueryRecord class
  QueryField collection
    See Also QueryRecord collection
  QueryList class *See Also* QueryList class
  QueryListValue class
    See Also QueryListValue class
  Query Manager 1885
  Query Metadata class
    See Also Query Metadata class
  Query Metadata collection
    See Also Query Metadata collection
  Query Monitor 1898
  QueryPrompt class
    See Also QueryPrompt class
  QueryPrompt collection
    See Also QueryPrompt collection
  QueryRecord class
    See Also QueryRecord class
  QueryRecord collection
    See Also QueryRecord collection
  QueryRecordHierarchy class
    See Also QueryRecordHierarchy class
  QueryRecordHierarchy collection
    See Also QueryRecordHierarchy collection
  QuerySecurityProfile class
    See Also QuerySecurityProfile class
  QuerySelect class *See Also* QuerySelect class
  QuerySelect collection
    See Also QuerySelect collection
  QueryStatistics class
    See Also QueryStatistics class
  recursive 1951
  running 1900
  SearchQuery class
    See Also SearchQuery class
  security 1901
  setting a drilling URL 1896
  setting the expression type 1894
  setting user languages 1901
  using criteria and expressions 1894
  using metadata 1898
QueriesPromptsArray property
  CONQRSMGR class 691
query API
  understanding 1891
QueryArray property
  CONQRSMGR class 692
Query class
  list of methods, properties, returns 2837
  methods *See Also* Query class methods
  properties *See Also* Query class properties
  understanding 1922
query classes
  adding criteria 2068
  collections in 1886
  creating queries 2066
  examples 2066
  handling errors 1901
  hierarchy 1888
  outer joins 2074
  Query class *See Also* Query class
  QueryCriteria *See Also* QueryCriteria class
  QueryDBRecord
    See Also QueryDBRecord class
  QueryDBRecordField
    See Also QueryDBRecordField class
  QueryExpression
    See Also QueryExpression class
  QueryField class *See Also* QueryField class
  query fields 1902
  QueryList *See Also* QueryList class
  QueryListValue
    See Also QueryListValue class
  Query Metadata
    See Also Query Metadata class
  QueryOutputFields 1902
  QueryPrompt *See Also* QueryPrompt class
  QueryRecord *See Also* QueryRecord class
  QueryRecordHierarchy
    See Also QueryRecordHierarchy class
  QuerySecurityProfile
    See Also QuerySecurityProfile class
  QuerySelect *See Also* QuerySelect class
  QuerySelectedFields 1902
  QueryStatistics
    See Also QueryStatistics class
  running queries 1900
  security 1901
  Session class methods
    See Also Session class methods
  understanding 1885, 1891
Query classes
  list of methods, properties, returns 2837
Query class methods
  AddPrompt 1922
  AddQuerySelect 1923
  AddTrackingURL 1924
  Close 1925
  CopyPrivateQuery 1925
  Create 1926
  Delete 1928
  DeletePrompt 1929
  FindExpression 1929
  FormatBinaryResultString 1930
  FormatResultString 1931

GetTreePromptCount 1932
Open 1933
Rename 1934
RunToFile 1935
RunToRowset 1938
RunToString 1941
Save 1944
SetTrackingURL 1944
Query class properties
Approved 1946
ApproveDtTm 1947
ApproveOprId 1946
ApproveUserId 1946
CreateDtTm 1947
CreateOprId 1947
CreateUserId 1947
Description 1948
Disabled 1948
ExecAppName 1948
ExecLogging 1949
Folder 1949
LastSQLErrorCode 1949
LastUpdDttm 1949
LastUpdOprId 1950
LongDescription 1950
Metadata 1950
MetaSQL 1950
MoreRowsAvailable 1951
Name 1951
OutputUnicode 1951
PDFFont 1952
PromptRecord 1953
Prompts 1953
PublicPrivate 1953
QuerySelect 1954
QueryStatistics 1954
RunTimePrompts 1954
SQL 1955
Type 1955
Query collection
list of methods, properties, returns 2837
methods *See Also* Query collection methods
properties 1921
understanding 1918
Query collection methods
First 1919
Item 1920
ItemByName 1920
Next 1921
Query collection properties 1921
query collections
Query *See Also* Query collection
QueryCriteria
See Also QueryCriteria collection
QueryDBRecord
See Also QueryDBRecord collection
QueryDBRecordField
See Also QueryDBRecordField collection
QueryExpression
See Also QueryExpression collection
QueryField *See Also* QueryField collection
Query Metadata
See Also Query Metadata collection
QueryPrompt
See Also QueryPrompt collection
QueryRecord
See Also QueryRecord collection
QueryRecordHierarchy
See Also QueryRecordHierarchy collection
QuerySelect *See Also* QuerySelect collection
understanding 1886
QueryCriteria
class *See Also* QueryCriteria class
collection *See Also* QueryCriteria collection
objects 1893, *See Also* QueryCriteria objects
QueryCriteria class
list of methods, properties, returns 2846
methods
See Also QueryCriteria class methods
properties
See Also QueryCriteria class properties
understanding 1998
QueryCriteria class methods
AddExpr1Expression 1998
AddExpr1Field 1999
AddExpr2Expression 2000
AddExpr2Field1 2000
AddExpr2Field2 2001
AddExpr2List 2002
AddExpr2Subquery 2002
QueryCriteria class properties
Expr1Expression 2003
Expr1Field 2003
Expr1Type 2004
Expr2Constant1 2005
Expr2Constant2 2005
Expr2Expression1 2005
Expr2Expression2 2006
Expr2Field1 2006
Expr2Field2 2006
Expr2List 2007
Expr2Subquery 2007
Expr2Type 2007
Logical 2011
LParenLvl 2011
Name 2011
Negation 2012
Operator 2012
R1ExprNum 2014
R1ExprType 2014
R2ExprNum 2014
R2ExprType 2015
RParenLvl 2015
QueryCriteria collection
list of methods, properties, returns 2846
methods
See Also QueryCriteria collection methods
properties 1997
understanding 1994
QueryCriteria collection methods
First 1995
Item 1995
ItemByName 1996
Next 1997
QueryCriteria objects 1893, 1998
QueryDBRecord
class *See Also* QueryDBRecord class
collection
See Also QueryDBRecord collection
objects 2045
QueryDBRecord class
list of methods, properties, returns 2853
methods
See Also QueryDBRecord class methods
properties 2047
understanding 2045

QueryDBRecord class methods
QueryDBRecordFieldByIndex 2045
QueryDBRecordFieldByName 2046
QueryDBRecord collection
list of methods, properties, returns 2853
methods
See Also QueryDBRecord collection methods
properties 2045
understanding 2042
QueryDBRecord collection methods
First 2042
Item 2043
ItemByName 2043
Next 2044
QueryDBRecordField
class *See Also* QueryDBRecordField class
collection
See Also QueryDBRecordField collection
objects 2052
QueryDBRecordFieldByIndex method 2045
QueryDBRecordFieldByName method 2046
QueryDBRecordField class
list of methods, properties, returns 2854
methods
See Also QueryDBRecordField class methods
properties
See Also QueryDBRecordField class properties
understanding 2052
QueryDBRecordField class methods
GetImageFormat 2052
QueryDBRecordField class properties
Decimal 2053
Description 2053
Flag 2054
Format 2055
Length 2056
LongName 2056
LookupTableName 2056
Name 2056
Shortname 2057
Type 2057
QueryDBRecordField collection
list of methods, properties, returns 2854
methods
See Also QueryDBRecordField collection methods
properties 2051
understanding 2048
QueryDBRecordField collection methods
First 2048
Item 2049
ItemByName 2049
Next 2050
Sort 2051
QueryDBRecordField collection properties 2051
QueryDBRecordField objects 2052
QueryDBRecordFields property 2047
QueryDBRecord objects 2045
QueryExpression
class *See Also* QueryExpression class
collection
See Also QueryExpression collection
objects 2018
QueryExpression class
list of properties, returns 2848
properties
See Also QueryExpression class properties
understanding 2018
QueryExpression class properties
Aggregate 2019
BindFlag 2019
Decimal 2020
ExpNum 2020
IsXlatExpression 2020
Length 2020
Name 2020
OutputField 2021
RightExprFlag 2021
SelectedField 2021
Text 2021
Type 2022
QueryExpression collection
list of methods, properties, returns 2848
methods
See Also QueryExpression collection methods
properties 2018
understanding 2015
QueryExpression collection methods
First 2015
Item 2016
ItemByName 2017
Next 2017
QueryExpression objects 2018
QueryField
class *See Also* QueryField class
collection *See Also* QueryField collection
objects 1984
QueryField class
list of methods, properties, returns 2844
methods *See Also* QueryField class methods
properties
See Also QueryField class properties
understanding 1984
QueryField class methods
AddTranslateExpression 1984
AddTranslateField 1985
GetImageFormat 1985
QueryField class properties
Aggregate 1986
ColumnNumber 1987
Decimal 1987
Description 1987
ExpNum 1987
Flag 1988
Format 1988
HeadingText 1989
HeadingType 1989
HeadingUniqueFieldName 1989
Length 1990
LongName 1990
Name 1990
OrderByDirection 1990
OrderByNumber 1991
QueryRecord 1991
RecordAlias 1992
ShortName 1992
TranslateEffDtLogic 1992
TranslateExpression 1993
TranslateField 1993
TranslateOption 1993
Type 1994
QueryField collection

list of methods, properties, returns 2843
methods
See Also QueryField collection methods
properties 1983
understanding 1980
QueryField collection methods
First 1980
Item 1981
ItemByExpNum 1982
ItemByNameAndAlias 1981
Next 1983
QueryField objects 1984
QueryFields property 1979
QUERYITEMPROMPT class
about 705
list of properties 2685
QUERYITEMPROMPT class properties
QueryName 705
QueryPromptRecord 705
QueryList class
list of methods, properties, returns 2849
methods *See Also* QueryList class methods
properties 2025
understanding 2023
QueryList class methods
AddListValue 2023
First 2024
Item 2024
Next 2025
QueryListValue class
list of properties, returns 2850
properties 2026
understanding 2026
Query Manager 1885
query metadata 1898
class *See Also* Query Metadata class
collection
See Also Query Metadata collection
objects 2035
Query Metadata class
list of properties, returns 2851
properties
See Also Query Metadata class properties
understanding 2035
Query Metadata class properties
Name 2035
Value 2036
Query Metadata collection
list of methods, properties, returns 2851
methods
See Also Query Metadata collection methods
properties 2034
understanding 2031
Query Metadata collection methods
First 2032
Item 2032
ItemByName 2033
Next 2034
query metadata objects 2035
Query Monitor 1898
QueryName property
QUERYITEMPROMPT class 705
query objects
declaring/scope 1903
QueryCriteria 1893
QueryDBRecordFields 1894
QueryDBRecords 1894
QueryOutputFields 1893
QueryRecords 1893
QuerySelectedFields 1893
understanding 1885, 1892, 1922
understanding QuerySelect 1892
QueryOutputFields property 1971
QUERYPARAMETER_CHILDFEEDID property
Utility class 891
QUERYPARAMETER_DATATYPEID property
Utility class 891
QUERYPARAMETER_DEFLOCALNODE property
Utility class 892
QUERYPARAMETER_DSSCOUNT property
Utility class 892
QUERYPARAMETER_DSSNAME property
Utility class 892
QUERYPARAMETER_DSSVALUE property
Utility class 892
QUERYPARAMETER_FEEDFORMAT property
Utility class 892
QUERYPARAMETER_FEEDID property
Utility class 893
QUERYPARAMETER_FEEDLIST property
Utility class 893
QUERYPARAMETER_FEEDTYPE property
Utility class 893
QUERYPARAMETER_IBTRANSID property
Utility class 893
QUERYPARAMETER_IFMODIFIEDSINCE property
Utility class 893
QUERYPARAMETER_IFNONEMATCH property
Utility class 894
QUERYPARAMETER_KEYWORD property
Utility class 894
QUERYPARAMETER_LANGUAGE property
Utility class 894
QUERYPARAMETER_NODENAME property
Utility class 894
QUERYPARAMETER_PAGENUM property
Utility class 894
QUERYPARAMETER_PORTALNAME property
Utility class 895
QUERYPARAMETER_PTPPB_SEARCH_MODE property
Utility class 895
QUERYPARAMETER_PTPPB_SEARCH_TEXT property
Utility class 895
QueryPrompt
class *See Also* QueryPrompt class
collection *See Also* QueryPrompt collection
QueryPrompt class
list of properties, returns 2856
properties
See Also QueryPrompt class properties
understanding 2061
QueryPrompt class properties
EditType 2061
FieldDecimal 2062
FieldFormat 2062
FieldLength 2063
FieldName 2063
FieldType 2063

HeadingText 2064
HeadingType 2064
LangCount 2065
Name 2065
PromptRecordFieldName 2065
PromptTable 2065
UniquePromptName 2066
UseCount 2066
QueryPrompt collection
list of methods, properties, returns 2855
methods
See Also QueryPrompt collection methods
properties 2061
understanding 2058
QueryPrompt collection methods
First 2058
Item 2059
ItemByName 2059
Next 2060
QueryPrompt collection properties 2061
QueryPromptRecord property
QUERYITEMPROMPT class 705
QueryRecord
class *See Also* QueryRecord class
collection *See Also* QueryRecord collection
objects 1976
QueryRecord class
list of methods, properties, returns 2843
methods 1977
properties
See Also QueryRecord class properties
understanding 1976
QueryRecord class methods 1977
QueryRecord class properties
Description 1978
JoinAlias 1978
JoinFieldName 1978
JoinType 1978
Name 1979
QueryFields 1979
RecordAlias 1979
QueryRecord collection
list of methods, properties, returns 2842
methods
See Also QueryRecord collection methods
properties 1976
understanding 1973
QueryRecord collection methods
First 1974
Item 1974
ItemByAlias 1975
Next 1975
QueryRecord collection properties 1976
QueryRecordHierarchy
class *See Also* QueryRecordHierarchy class
collection
See Also QueryRecordHierarchy collection
objects 2027
QueryRecordHierarchy class
list of properties, returns 2850
properties
See Also QueryRecordHierarchy class properties
understanding 2030
QueryRecordHierarchy class properties
Description 2030
Level 2031
Name 2031
ParentFlag 2031
QueryRecordHierarchy collection
list of methods, properties, returns 2850
methods
See Also QueryRecordHierarchy collection methods
properties 2030
understanding 2027
QueryRecordHierarchy collection methods
First 2027
Item 2028
ItemByName 2028
Next 2029
QueryRecordHierarchy objects 2027, 2030
QueryRecord objects 1976
QueryRecord property 1991
QueryRecords property 1972
QuerySecurityProfile class
list of properties, returns 2852
properties
See Also QuerySecurityProfile class properties
understanding 2037
QuerySecurityProfile class properties
AllowAnyJoin 2038
AllowDistinct 2038
AllowExpressions 2038
AllowSubqueries 2038
AllowUnions 2038
ApprovePrivateQuery 2039
ApprovePublicQuery 2039
CanCreatePublic 2039
CanCreateWorkFlow 2039
CanModifyQuery 2040
CanRunQuery 2040
CanRunToCrystal 2040
CanRunToExcel 2040
LimitUnapproved 2041
MaxInTreeCriteria 2041
MaxJoins 2041
MaxRowsToFetch 2041
MaxUnapprovedRows 2042
QuerySelect
class *See Also* QuerySelect class
collection *See Also* QuerySelect collection
objects 1960
understanding select statements 1892
QuerySelect class
list of methods, properties, returns 2840
methods *See Also* QuerySelect methods
properties
See Also QuerySelect class properties
understanding 1960
QuerySelect class properties
Criteria 1970
Distinct 1970
Expressions 1970
HavingCriteria 1971
ParentSelectNum 1971
QueryOutputFields 1971
QueryRecords 1972
QuerySelectedFields 1972
QuerySelects 1972
SelectNum 1973
SelectType 1973
QuerySelect collection
list of methods, properties, returns 2842
methods

See Also QuerySelect collection methods
properties 1960
understanding 1956
QuerySelect collection methods
AddUnion 1956
DeleteQuerySelect 1957
First 1957
Item 1958
ItemBySelNum 1958
Next 1959
QuerySelectedFields property 1972
QuerySelect methods
AddAllFields 1961
AddCriteria 1962
AddExpression 1963
AddHavingCriteria 1963
AddQueryOutputField 1964
AddQueryRecord 1965
AddQuerySelectedField 1965
DeleteCriteria 1966
DeleteExpression 1967
DeleteField 1968
DeleteHavingCriteria 1968
DeleteRecord 1969
QuerySelect objects 1960
QuerySelect property 1954
QuerySelects property 1972
QueryStatistics class
list of properties, returns 2851
properties
See Also QueryStatistics class properties
understanding 2036
QueryStatistics class properties
AvgExecTime 2036
AvgFetchTime 2036
AvgNumRows 2037
ExecCount 2037
LastExecDtTm 2037
QueryStatistics property 1954
QueryString property 1071
QueryText property 2480
QueueName property 1307
queues, universal *See* universal queues
QueueSeqId property 1308

R

R1ExprNum property 2014
R1ExprType property 2014
R2ExprNum property 2014
R2ExprType property 2015
RangeFrom property 2318
Range property
DataSourceParameter class 839
DataSourceSetting class 851
RangeTo property 2319
rating box chart 425
example 592
methods 543
properties 547
RatingBoxChart class
list of methods 2679
list of properties 2679
methods
See Also RatingBoxChart class methods
objects 430
properties
See Also RatingBoxChart class properties
RatingBoxChart class methods 543
SetLegendImg 544
SetRBNodeData 544
SetRBNodeRecord 545
SetXAxisLabels 545
SetYAxisLabels 546
RatingBoxChart class properties
BoxMaxDisplayItems 547
DraggedNodeStyle 547
HasLegend 548
Height 548
IconOnlySelectedQuadrantStyle 549
IsDragable 549
LegendPosition 549
MainTitle 550
MainTitleStyle 550
NDMaxDisplayDescLength 550
PopUpHeaderStyle 551
PopUpHeight 551, 555
PopUpStyle 552
PopUpWidth 552
SelectedQuadrantStyle 552
ShowNodeDescription 553
Style 553
ViewAllStyle 553
Width 554
XAxisBoxNum 554
XAxisLabelStyle 554
XAxisTitle 555
XAxisTitleStyle 555
YAxisLabelStyle 556
YAxisTitle 556
YAxisTitleStyle 556
read-only properties 215
readable property 187
ReadAllEmailHeadersWithAttach method 1134
ReadBlobFromFile method 1409
ReadEmails method 1135
ReadEmailsWithAttach method 1136
ReadEmailsWithUID method 1137
ReadEmailWithAttach method 1138
ReadFromDatabase method 1151
ReadHeaders method 1139
ReadLine method 993
ReadOnce property 187
ReadRecord method 1020
ReadRowset method 994, 1025
Reassign method
Worklist class 1467
WorklistEntry class 1470
Recalculate method 34
Recipients property
MCFEmail class 1130
MCFOutboundEmail class 1197
recname property 2123
RecName property 2498
RecOpts property 2496
RecordAlias property
QueryField class 1992
QueryRecord class 1979
Record class
list of functions, methods, properties, returns 2762
methods *See Also* Record class methods
properties *See Also* Record class properties
shortcut considerations 2078

understanding 2077
using SQL 2241
Record class methods
CompareFields 2080
CopyChangedFieldsTo 2081
CopyFieldsTo 2082
creating SQL statements 2078
DBPatternMatch 2084
Delete 2085
ExecuteEdits 2086, 2101
GetField 2089
Insert 2090
Save 2092
SearchClear 2095
SelectByKey 2096
SelectByKeyEffDt 2098
SetDefault 2099
SetEditTable 2087, 2100
Update 2102
Record class properties
FieldCount 2104
fieldname 2104
IsChanged 2105
IsDeleted 2105
IsEditError 2106
Name 2106
ParentRow 2107
RelLangRecName 2107
SystemIDFieldName 2108
TimeStampFieldName 2109
RecordCount property 2123
RecordHierachy property 2047
RecordName property
cube collection 123
SearchRecordField class 2505
record objects
declaring/scope 2079
understanding 2077, 2239
records
AdvancedSearchRecords method 1908
callback 1543
case sensitivity 2240
editing via ExecuteEdits/SetEditTable 1269
FindQueryDBRecords method 1910
objects *See Also* record objects
PT_DETMSGS 1501
PT_OPTDETMSGS 1496
PT_OPTPARMS
See Also PT_OPTPARMS record
publishing/subscribing to partial 1251
QueryDBRecords 1894
query definitions 1893
Record class
See Also Record class, Record class
sorting 2136
understanding 2077, 2239
understanding work records 383
recursive queries 1951
RedirectURL method 1079
referencing
referencing PortalRegistry objects via C/C++ 1820
referencing PortalRegistry objects via Visual Basic 1820
self-referencing 210
superclass referencing 218
using arrays with object references 274
RefIDs property
MCFEmail class 1130
MCFOutboundEmail class 1198
RefreshDescription method 2313, 2344
refreshing
PhysicalQueue class Refresh method 2447
PhysicalQueue class RefreshTaskList method 2448
Task class Refresh method 2456
Task class RefreshStatus method 2456
Refresh method
Agent class 2435
AgentPhysQueueTasks class 2441
chart data 432, 477
PhysicalQueue class 2447
Rowset class 2153
Task class 2456
TaskList class 2459
RefreshOnChange property
DataSourceSetting class 851
RefreshQTaskList method 2435
RefreshStatus method 2456
RefreshTaskList method 2448
regional settings
class *See Also* Regional Settings class
objects 2182
RegionalSettings property 2189
Regional Settings class
list of properties, returns 2790
properties
See Also Regional Settings class properties
Regional Settings class properties
1159Separator 2202
2359Separator 2203
ClientTimeZone 2198
CurrencyFormat 2199
CurrencySymbol 2199
DateFormat 2199
DateSeparator 2200
DecimalSymbol 2200
DigitGroupingSymbol 2200
LanguageCD 2201
UseLocalTime 2202
regional settings objects 2182
RegionalSettings property 2189
RegisteredBy property
CONQRSMGR class 692
RegisteredDTTM property
CONQRSMGR class 692
registry, portal *See* portal registry
RejectTransform property 2288
RelatedFieldValue property
DataSourceSetting class 851
relative paths
ReadBlobFromFile method 1410
using 982
WriteBlobToFile method 1416
RelativeURL property
Content Reference class 1693
ContentReference links 1730
Request class 1071
RelLangRecName property 2107
RemoteAddr property 1071
RemoteFrameworkURL property 1384
RemoteHost property 1071
RemoteNode collection
list of methods, properties, returns 2813
methods
See Also RemoteNode collection methods

properties 1654
understanding 1652
RemoteNode collection methods
First 1652
ItemByName 1652
Next 1653
RemoteUser property 1072
RemoveAllChildNode method 2586
RemoveChildNode method 2587
RemoveEmail method 1140
RemoveEmails method 1141
Remove method 2321
RemoveUser method 1239
RenameCubeCollection method 85
RenameCube method 84
RenameDimension method 85
RenameExplicitDimensionSet method 86
RenameMember method 35
Rename method
AnalyticModelDefn class 83
analytic type definition 181
Node class 2344
Query class 1934
Tree class 2376
Tree Structure class 2406
RenameOrganizer method 87
RenameUserFunction method 88
RENURLID property 2452
Replace method 275
ReplyIDs property
MCFEmail class 1130
MCFOutboundEmail class 1198
ReplyTo property
MCFEmail class 1130
MCFOutboundEmail class 1198
Report class
about 2613
import statements 2614
Report class methods
constructor 2614
Report 2614
Report class properties
contentId 2615
CreatedDateTime 2615
DatabaseName 2615
Description 2616
ExpireDate 2616
FolderName 2616
ProcessInstanceID 2617
ReportInstanceID 2617
ReportName 2617
ReportURL 2617
ReportDefn class
about 2600
import statements 2600
ReportDefn class methods
Close 2601
constructor 2600
DisplayOutput 2602
Get 2602
GetOutDestFormatString 2603
GetPSQueryPromptRecord 2604
PrintOutput 2604
ProcessReport 2605
Publish 2606
ReportDefn 2600
SetPSQueryPromptRecord 2607
SetRuntimeDataXMLFile 2608
SetRuntimeProperties 2609
ReportDefn class properties
Description 2611
DestinationPrinter 2611
FolderName 2611
OutDestination 2611
OutDestinationType 2612
ProcessInstance 2612
Status 2613
UseBurstValueAsOutputFileName 2613
ReportDefn method
ReportDefn class 2600
ReportDescr property 1830
ReportFolder property 1831
ReportId property 1831
ReportInstanceID property 2617
ReportManager class
about 2618
import statements 2618
ReportManager class methods
constructor 2618
ReportManager 2618
report manager definition classes 2600
about 2596
ReportManager method
ReportManager class 2618
ReportManager methods
AddSearchFieldCriteria 2619
GetReportList 2620
SetBurstFieldCritera 2620
SetCaseSensitive 2621
SetDateCriteria 2622
SetFolderCriteria 2623
SetProcessInstanceCriteria 2623
SetReportIDCriteria 2624
SetUserIdCriteria 2624
report manager search classes
about 2596, 2613
Report method
Report class 2614
ReportName property 2617
reports
ReportFolder property 1831
ReportId property 1831
ReportURL property 2617
Repository class
list of properties, returns 2793
properties
See Also Repository class properties
Repository class properties
Bindings 241
Namespaces 241
Repository property 2190
Request class
list of methods, properties, returns 2721
methods *See Also* Request class methods
mobile PeopleCode 1442
properties *See Also* Request class properties
understanding 1060
Request class methods
GetContentBody method 1062
GetCookieNames 1063
GetCookieValue method 1064
GetHeader 1064
GetHeaderNames 1064
GetHelpURL method 1065
GetParameter method 1065
GetParameterNames method 1065

GetParameterValues method 1066
Request class properties
AuthTokenDomain 1066
AuthType 1067
BrowserPlatform 1067
BrowserType 1067
BrowserVersion 1068
ByPassSignOn 1067
ContentURI 1068
ExpireMeta 1068
FullURI 1069
HTTPMethod 1069
LogoutURL 1070
PathInfo 1070
Protocol 1070
QueryString 1071
RelativeURL 1071
RemoteAddr 1071
RemoteHost 1071
RemoteUser 1072
RequestURI 1072
Scheme 1073
ServerName 1073
ServerPort 1073
ServletPaty 1074
Timeout 1074
RequestedFields property 2480
request handlers
implementing 930
RequestInfo property
Utility class 895
RequestingNodeDescription property 1370
RequestingNodeName property 1370
requestmessageid property 1479
request messages
creating 1497
editing PeopleCode 1502
getting the trace level 1500
setting the trace level 1499
understanding 1498
request objects
accessing 1059
declaring 1060
understanding 1060
requests
%Request system variable 1059
class *See Also* Request class
objects *See Also* request objects
RequestURI property 1072
Required property 652
DataSourceParameter class 839
DataSourceSetting class 852
reserved internal functions
Generate method 243
reserved words 2869
ResetCursor method 356
resetEntries method
FeedDoc class 909
resetFeedAttributes method
Feed class 783
resetFeedSecurities method
Feed class 784
Reset method
chart data 478
resetParameters method
DataSource class 825
ResetQueriesPrompt method
CONQRSMGR class 677
resetUserValues method
DataSourceParameter class 834
ResolveCircularDeps property 93
ResponseAsAttachment property 1370
Response class
list of methods, properties, returns 2723
methods *See Also* Response class methods
properties 1082
understanding 1061
Response class methods
Clear 1075
CreateCookie 1075
GetCookie 1075
GetCookieNames 1076
GetHeader 1076
GetHeaderNames 1076
GetImageURL 1077
GetJavaScriptURL 1077
GetStyleSheetURL 1078
RedirectURL 1079
SetContentType 1079
SetHeader 1080
UseSimpleURL 1080
Write 1081
WriteLine 1081
response messages
building 1507, 1508
creating 1501
editing PeopleCode 1506
understanding 1501
response objects
accessing 1059
declaring 1060
understanding 1061
responses
%Response system variable 1059
class *See Also* Response class
objects *See Also* response objects
Responses property 1465
Responses property 1465
ResponseStatus property 1308, 1479
RestoreBounds method 1573
Resubmit method 1328
ResultOfSend property 1198
Result property 727
RetrieveEmail method 1159
return receipts *See* email
ReuseCursor property 2244, 2256
Reverse method 277
RevertToPre850 property 463, 515
RevokePermissionForComponent method 1636
RevokePermissionForScript method 1637
RightExprFlag property 2021
RolePermissions property
Content Reference class 1694
ContentReference links 1731
Folder class 1673
RolePermissionValue
collection
See Also RolePermissionValue collection
RolePermissionValue collection
methods
See Also RolePermissionValue collection methods
properties 1721
understanding 1717
RolePermissionValue collection methods
DeleteItem 1717

First 1718
InsertItem 1719
ItemByName 1719
Next 1720
RolePermissionValue Collection methods 2822
RootElement property
FeedDoc class 917
RootFolder property 1642
RotationAngle property 463
RoutingDefnName property 2289
Row class
list of functions, methods, properties, returns 2763
methods *See Also* Row class methods
properties *See Also* Row class properties
shortcut considerations 2112
understanding 2111
Row class methods
CopyTo 2113
GetNextEffRow 2114
GetPriorEffRow 2114
GetRecord 2115
GetRowset 2117
scrollname 2118
Row class properties
ChildCount 2119
DeleteEnabled 2119
IsChanged 2120
IsDeleted 2121
IsEditError 2121
IsNew 2122
ParentRowset 2122
recname 2123
RecordCount 2123
RowNumber 2123
Selected 2124
Style 2124
Visible 2125
RowCount property 2168
RowFetch property 2208
RowNumber property 2123
row objects
declaring/scope 2112
understanding 2111
Row property
AssignedPagelet class 1750
AvailablePagelet class 1760
SelectedPagelet class 1798
rows
effective-dated 2149
objects *See Also* row objects
Row class *See Also* Row class
RowsAffected property 2257
rowset-based messages 1245
RowsetCache class 2171
data type 2173
error handling 2172
list of functions, methods, properties, and returns 2768
properties
See Also RowsetCache class properties
scope 2173
understanding 2171
using 2172
RowsetCache Class
methods
See Also RowsetCache class methods
RowsetCache class methods
Delete 2173
Get 2174
Save 2175
RowsetCache class properties
Content 2177
Description 2177
Rowset class
list of functions, methods, properties, and returns 2765
methods *See Also* Rowset class methods
properties *See Also* Rowset class properties
shortcut considerations 2128
understanding 2127
Rowset class methods
ClearDeletesChanges 2129
CopyTo 2131
DeleteRow 2133
Fill 2135
FillAppend 2138
Flush 2140
FlushRow 2142
GetCurrEffRow 2143
GetFirstUserSortedRow 2143
GetLastUserSortedRow 2144
GetNewEffRow 2145
GetRow 2146
HideAllRows 2147
InsertRow 2148
IsUserSorted 2149
MapBufRowToUserSortRow 2150
MapUserSortRowToBufRow 2151
Refresh 2153
Select 2154
SelectNew 2156
SetDefault 2158
ShowAllRows 2159
Sort 2160
Rowset class properties
ActiveRowCount 2161
ChangeOnInit 2162
DataAreaCollapsed 2162
DBRecordName 2163
DeleteEnabled 2163
EffDt 2164
EffSeq 2165
InsertEnabled 2165
IsEditError 2166
Level 2167
Name 2167
ParentRow 2168
ParentRowset 2168
RowCount 2168
SetComponentChanged 2169
TopRowNumber 2169
rowset objects
declaring/scope 2128
understanding 2127
rowsets
Business Interlink objects 305
component buffer 1248
message 1248
objects *See Also* rowset objects
RowsetCache class
See Also RowsetCache class
Rowset class *See Also* Rowset class
structure for email 1108
transforming XML 2532
understanding 2127

understanding charts 395
rowsets, file 1026
RParenLvl property 2015
Rte 1458
RuleDefn class
AddRuleExpression method 129
methods 129
properties 131
RuleString property 131
RuleExpressions
classes 131
using constant class 132
rules 103, 105, 123, 124
adding
RuleDefn class 129
ExpressionBlock class 142
generating
RuleDefn class 130
assignment class 132
comparison class 134
constant class 137
cube class 140
FunctionCall class 147
MemberReference class 149
Operation class 151
variable class 154
getting
ExpressionBlock class 142
RuleString property 131
RUN_CNTL_ID property
SCHED_INFO class 708
RunAsynch method
running optimization transactions 1492
understanding 1534
RunAsync method 22
RunCntlId_Auto property
CONQRS_CONST class 700
RunCntlId property 2513
CONQRSMGR class 692
RunControlID property 1874
RunControlParArray property
CONQRSMGR class 693
RunDateTime property 1874
RunJobSetNow method 1855
RunLocation property 1875
Run method
CONQRSMGR class 677
RunMode_Prev_DefRowNumber property
CONQRS_CONST class 701
RunMode_Prev property
CONQRS_CONST class 701
RunMode_Sample property
CONQRS_CONST class 702
RunMode_Sched_File property
CONQRS_CONST class 702
RunMode_Sched_Web property
CONQRS_CONST class 702
RunMode_Window property
CONQRS_CONST class 703
RunMode_XMLFormattedString property
CONQRS_CONST class 703
RunMode property
CONQRSMGR class 693
RunRecurrence property
ProcessRequest class 1875
Search Schedule class 2514
RunStatus property 1875
RunSynch method
running optimization transactions 1492
understanding 1536
RunSync method 23
RunTimePrompts property 1954
RunToFile method 1900, 1935
RunToRowset method 1887, 1900, 1938
RunToString method 1941
RunToWindow method
CONQRSMGR class 678
RunToXMLFormattedString method
CONQRSMGR class 679

S

SameTime instant messaging 1233
SaveAsDraft method 2379
saveAs method
Feed class 785
SaveAs method 2378
SaveDraft method 2381
save method
Feed class 785
Save method
AESection class 8
AnalyticModelDefn class 89
analytic type definition 182
Component Interface class 630
CONQRSMGR class 680
Content Reference class 1684
ContentReference links 1722
Favorite collection 1807
Folder class 1663
Mobile class 1410
PageletCategory class 1767
Pagelet class 1777
Portal class 1654
PortalRegistry class 1638
Query class 1944
Record class 2092
RowsetCache class 2175
SearchIndex class 2494
TabDefinition class 1737
Tree class 2377
Tree Structure class 2406
UserHomepage class 1789
WorklistEntry class 1471
SaveRunControlParms method
CONQRSMGR class 681
SaveToDatabase method 1151
SaveWithCustomData method 1472
ScenarioManaged property 187
SCHED_INFO class
about 706
list of properties 2685
SCHED_INFO class properties
AE_ID 706
DIRLOCATION 707
OPRID 707
OUTDESTTYPE 707
PRCSFILENAME 707
PROCESS_INSTANCE 708
RUN_CNTL_ID 708
SchedInfo property
CONQRSMGR class 695
SchedRequest_CQR property
CONQRS_CONST class 704

SchedRequest_XMLP property
CONQRS_CONST class 704
SchedRequestType property
CONQRSMGR class 696
Schedule method 1855
schedules, search *See* search schedules
Schedules property 2496
schema
messages 1321, 1326
Scheme property 1073
ScoreAsNumber property 2488
ScorePrecision property 2481
Score property 2487
scrollname method 2118
SearchAttribute class
about 2625
import statements 2625
SearchAttribute class methods
constructor 2625
SearchAttribute 2625
SearchAttribute method
SearchAttribute class 2625
SearchAttribute properties
attrName 2626
attrValue 2626
compareOp 2626
FileURL 2616
Search classes
examples 2525
general purpose search 2525
handling errors 2470
hierarchy 2468
ParseResult *See Also* ParseResult class
portal search API 2526
SearchField *See Also* SearchField class
Search FS Options
See Also Search FS Options class
Search HTTP Options
See Also Search HTTP Options class
SearchIndex *See Also* SearchIndex class
Search Language
See Also Search Language class
SearchQuery *See Also* SearchQuery class
SearchRecordField
See Also SearchRecordField class
SearchRecord Options
See Also SearchRecord Options class
SearchResult *See Also* SearchResult class
Search Schedule
See Also Search Schedule class
Search Start Options 2524
SearchClear method
Field class 944
Record class 2095
Search collections
ParseResult *See Also* ParseResult collection
SearchField *See Also* SearchField collection
SearchIndexes
See Also SearchIndexes collection
Search Language
See Also Search Language collection
SearchRecordField
See Also SearchRecordField collection
SearchResult
See Also SearchResult collection,
SearchResult collection
Search Schedule
See Also Search Schedule collection
Search Start Options
See Also Search Start Options collection
SearchDefault property
Field class 966
using with SearchClear property 2095
SearchEdit property 966
searches
AdvancedSearchQueries method 1905
AdvancedSearchRecords method 1908
languages *See Also* Search Language
objects 2471
results *See Also* SearchResult
schedules *See Also* search schedules
search collection path 2474
search indexes *See Also* search indexes
SearchIndex objects 2465
understanding 2465
Verity search classes
See Also Verity search classes
SearchField
class *See Also* SearchField class
collection *See Also* SearchField collection
objects 2491
SearchField class
list of properties, returns 2803
properties 2491
understanding 2491
SearchField collection
list of methods, properties, returns 2802
methods
See Also SearchField collection methods
properties 2490
understanding 2488
SearchField collection methods
First 2489
ItemByName 2489
Next 2490
SearchField collection properties 2490
SearchField objects 2491
SearchFields property 2488
Search FS Options
class *See Also* Search FS Options class
objects 2518
SearchFSOptions class
list of properties, returns 2808
Search FS Options class
properties
See Also Search FS Options class
properties
understanding 2518
Search FS Options class properties
changing values 2518
GlobList 2518
GlobListType 2519
MIMEList 2519
MIMEListType 2520
StartOpts 2520
Search FS Options objects 2518
Search HTTP Options
class *See Also* Search HTTP Options class
objects 2514
SearchHTTPOptions class
list of properties, returns 2807
Search HTTP Options class
properties
See Also Search HTTP Options class
properties
understanding 2514

Search HTTP Options class properties
AllowHTTPS 2515
changing values 2514
DomainLimit 2515
GlobList 2515
GlobListType 2515
LinkDepth 2516
MIMEList 2516
MIMEListType 2517
ProxyHost 2517
ProxyPort 2517
StartOpts 2518
Search HTTP Options objects 2514
SearchIndex class
list of, methods, properties, returns 2803
methods 2494
properties
See Also SearchIndex class properties
understanding 2493
SearchIndex class method
Save 2494
SearchIndex class properties
changing values 2494
ExtraOptions 2495
FSOpts 2495
HTTPOpts 2495
Languages 2495
Location 2496
Name 2496
RecOpts 2496
Schedules 2496
Type 2497
search indexes
class *See Also* SearchIndex class
collection *See Also* SearchIndexes collection
objects 2465
understanding 2465, 2493
SearchIndexes collection
list of methods, properties, returns 2803
methods
See Also SearchIndexes collection methods
properties 2493
understanding 2491
SearchIndexes collection methods
First 2492
ItemByName 2492
Next 2493
SearchIndex objects 2465
Search Language
class *See Also* Search Language class
collection
See Also Search Language collection
objects 2509
SearchLanguage class 2806
Search Language class
properties 2509
understanding 2509
SearchLanguage collection 2805
Search Language collection
methods
See Also Search Language collection methods
properties 2508
understanding 2505
Search Language collection methods
DeleteItem 2506
First 2506
InsertItem 2507
ItemByName 2507
Next 2508
Search Language objects 2509
search objects 2471
SearchPrivateQueries method 1915
SearchPublicQueries method 1917
search queries
class *See Also* SearchQuery class
objects 2475
PortalRegistry class GetSearchQuery method 2474
SearchQueryDBRecords method 1914
SearchQuery objects 2472, 2473, 2475
Session class GetSearchQuery method 2473
SearchQuery class
list of methods, properties, returns 2800
methods
See Also SearchQuery class methods
properties
See Also SearchQuery class properties
understanding 2475
SearchQuery class methods
Execute 2476
Parse 2476
SearchQuery class properties
HitCount 2478
Indexes 2478
IndexName 2479
KnowledgeBase 2480
Language 2480
ProcessedCount 2480
QueryText 2480
RequestedFields 2480
ScorePrecision 2481
SortSpecifications 2481
SearchQueryDBRecords method 1914
SearchQuery objects 2472, 2473, 2475
SearchRecordField
class *See Also* SearchRecordField class
collection
See Also SearchRecordField collection
objects 2504
SearchRecordField class
list of properties, returns 2805
properties
See Also SearchRecordField class properties
understanding 2504
SearchRecordField class properties
changing values 2504
FieldName 2504
IsAttachment 2504
IsVerityField 2505
IsWordIndex 2505
RecordName 2505
SearchRecordField collection
list of methods, properties, returns 2804
methods
See Also SearchRecordField collection methods
properties 2503
understanding 2500
SearchRecordField collection methods
DeleteItem 2501
First 2501
InsertItem 2502
ItemByName 2502

Next 2503
SearchRecordField objects 2504
SearchRecord Options
class *See Also* SearchRecord Options class
objects 2497
SearchRecord Options class
list of properties, returns 2804
properties
See Also SearchRecord Options class properties
understanding 2497
SearchRecord Options class properties
changing values 2497
Fields 2498
Filter 2498
IncrementalView 2498
RecName 2498
VeggieKey 2499
ZoneOptions 2500
SearchRecord Options objects 2497
SearchResult
class *See Also* SearchResult class
collection
See Also SearchResult collection, SearchResult collection
objects 2487
understanding 2467
SearchResult class
list of properties, returns 2802
properties
See Also SearchResult class properties
understanding 2487
SearchResult class properties
Key 2487
Score 2487
ScoreAsNumber 2488
SearchFields 2488
SearchResult collection
list of methods, properties, returns 2801
methods 2485
properties 2486
understanding 2467, 2485
using 2466
SearchResult objects 2487
SearchSchedule class 2807
Search Schedule class
properties
See Also Search Schedule class properties
understanding 2513
Search Schedule class properties
BuildType 2513
changing values 2513
RunCntlId 2513
RunRecurrence 2514
ServerName 2514
SearchSchedule collection 2806
Search Schedule collection
methods
See Also Search Schedule collection methods
properties 2512
understanding 2509
Search Schedule collection methods
DeleteItem 2510
First 2510
InsertItem 2511
ItemByName 2511
Next 2512
search schedule objects 2513
search schedules
class *See Also* Search Schedule class
collection
See Also Search Schedule collection
objects 2513
Search Start
objects 2524
options *See Also* Search Start Options
Search Start objects 2524
Search Start Options
class 2524
collection
See Also Search Start Options collection
SearchStartOptions class 2809
Search Start Options class 2524
SearchStartOptions collection 2808
Search Start Options collection
methods
See Also Search Start Options collection methods
properties 2523
understanding 2520
Search Start Options collection methods
DeleteItem 2521
First 2521
InsertItem 2522
ItemByName 2522
Next 2523
SEC_PROFILE class
about 708
import statements 709
list of properties 2686
SEC_PROFILE class methods
constructor 709
SEC_PROFILE 709
SEC_PROFILE class properties
CanCreatePublic 709
CanModify 709
CanRunQuery 709
SEC_PROFILE method
SEC_PROFILE class 709
SecProfile property
CONQRSMGR class 696
Secure property 1084
security
accessing PeopleSoft 2180
adding for iScripts 1054
file objects 977
queries 1901
QuerySecurityProfile class
See Also QuerySecurityProfile class
understanding PortalRegistry 1589
using for PortalRegistry classes 1591
SegmentCount property 1308
segments
deleting 1316
SegmentsByDatabase property 1308
SegmentsUnOrder property 1371
SelectAll method 2276
SelectByKeyEffDt method 2098
SelectByKey method
Record class 2096
WorklistEntry class 1473
SelectedField property 2021
SelectedPagelet
class *See Also* SelectedPagelet class
collection

See Also SelectedPagelet collection
SelectedPagelet class
list of properties, returns 2835
properties
See Also SelectedPagelet class properties
understanding 1796
SelectedPagelet class properties
CategoryName 1797
Column 1797
IsMinimized 1797
PageletName 1798
Row 1798
SelectedPagelet collection
list of methods, properties, returns 2835
methods
See Also SelectedPagelet collection methods
properties 1801
understanding 1798
SelectedPagelet collection methods
DeleteItem 1798
First 1799
InsertItem 1800
ItemByName 1800
Next 1801
SelectedPagelets property 1792
Selected property 2124
SelectedQuadrantStyle property 552
Select method 2154
SelectNew method 2156
SelectNum property 1973
SelectType property 1973
self-referencing 210
SendAll method 1206
Sender property
MCFEmail class 1131
MCFOutboundEmail class 1199
Send method 1205
MCFOutboundEmail class 1184
Notification class 1453
Sensitivity property
MCFEmail class 1131
MCFOutboundEmail class 1199
SequenceNumber property
Collection class 760
Compound class 753
Content Reference class 1694
ContentReference links 1731
Favorite class 1803
Folder class 1673
PageletCategory class 1773
Pagelet class 1783
Primitive class 747
TabDefinition class 1744
UserTab class 1792
ServerName property
PostReport class 1832
Request class 1073
Search Schedule class 2514
ServerPort property 1073
Server property
MCFInboundEmail 1156
SMTPSession class 1211
servers
application *See Also* application servers
mail *See Also* mail servers
PostReport class ServerName property 1832
Request class ServerName property 1073
Search Schedule class ServerName property 2514
ServerName property 1073
synchronization
See Also synchronization server
service operations 1246
Servlet property 1074
session
class *See Also* Session class
objects *See Also* session objects
Session class
API instantiation methods 2186
list of functions, methods, properties, returns, variables 2787
methods *See Also* Session class methods
properties *See Also* Session class properties
security 2180
understanding 2179
Session class methods
AdvancedSearchQueries 1905
AdvancedSearchRecords 1908
Connect 2184
Disconnect 2186
FindCompIntfcs 612
FindPortalRegistries 1611
FindQueries 1910
FindQueriesDateRange 1911
FindQueryDBRecords 1910
GetActualRemoteNodes 1612
GetCompIntfc 613, 1398
GetLocalNode 1613
GetNodes 1613
GetPortalRegistry 1614
GetQuery 1912
GetQuerySecurityProfile 1913
GetRemoteNodes 1615
GetSearchIndexes 2472
GetSearchQuery 2473
GetTree 2301
GetTreeStructure 2301
SearchPrivateQueries 1915
SearchPublicQueries 1917
SearchQueryDBRecords 1914
Session class properties
ErrorPending 2188
PSMessages 2188
PSMessagesMode 2188
RegionalSettings 2189
Repository 2190
SuspendFormatting 2190
TraceSettings 2190
WarningPending 2190
SessionID property 2280
session objects
declaring 2183
handling errors 2180
scope 2184
state considerations 2183
understanding 2179
understanding regional settings 2182
SetActualEndDate method 478
SetActualStartDate method 479
SetActualTaskBarColor method 480
SetAggregateMapping method 115
SetAnalyticInstance method 165
SetAttachmentContent method 1122
SetAttachmentProperty method 1360
setAttribute method

Feed class 786
SetBurstFieldCriteria method 2620
SetCaseSensitive method 2621
SetChartArea method 480
SetColorArray methods 433
SetComponentChanged property
Field class 967
Rowset class 2169
SetContentType method 1079
SetCrumbData method 519
SetCrumbRecord method 520
SetCursorPos method 945
SetDataAnnotations method 437
SetDataColor methods 438
SetDataGlyphScale 438
SetDataHints method 439
SetData method 45, 435
SetDataSeries method 443
setDataSourceById method
Feed class 787
SetDataURLs method 443
SetDataXAxis method 444
SetDataYAxis method 446
SetDateCriteria method 2622
SetDayFormat method 481
SetDefault method
Field class 946
Field class methods 945
Record class 2099
Rowset class 2158
SetDimensionAggregate method 104
SetDimensionOrder method 46
SetDimFilter method 46
SetDimSort method
CubeCollection class 47
CubeCollectionDefn class 116
SetDocumentKey method
DocumentKey class 739
SetDropdownData method 520
SetDropdownRecord method 521
SetEditTable method
Message class 1286
Message class methods 1269
Record class 2100
Record class methods 2087
SetEmailOption method 1857
SetFieldMapping method 117
SetFileId method 996
SetFileLayout method 998
SetFilter method 118
SetFolderCriteria method 2623
SetHeader method 1080
SetHourFormat method 482
SetID property 2400
SetImportMode method 2382
SetItemFolder method 1858
Set keyword 215
Set language construct 231
SetLayout method 166
SetLegendImg method 522, 544
SetLegend method 446, 483, 521, 543
SetMCFEmail method 1142
SetMessageError method 1329
setMessageHeadersAndMimeType method
Utility class 867
Set method 278
SetMinuteFormat method 484
SetModified method 1412
SetMonthFormat method 485
SetNodeData method 523
SetNodeRecord method 524
setNodeValue method
Utility class 868
SetNotifyAppMethod method
ProcessRequest class 1859
SetNotifyService method
ProcessRequest class 1862
SetOLDataAnnotations 448
SetOLDataGlyphScale 449
SetOLData method 447
SetOLDataSeries method 450
SetOLDataXAxis method 450
SetOLDataYAxis method 451
SetOutputOption method 1863
SetOutputParmDateArray method 1557
SetOutputParmDate method 1557
SetOutputParmDateTimeArray method 1559
SetOutputParmDateTime method 1558
SetOutputParmIntArray method 1561
SetOutputParmInt method 1561
SetOutputParmNumberArray method 1560
SetOutputParmNumber method 1559
SetOutputParmStringArray method 1563
SetOutputParmString method 1562
SetOutputParmTimeArray method 1564
SetOutputParmTime method 1563
SetParameter method 725
SetPersistAggregate method 119
SetPlannedEndDate method 486
SetPlannedStartDate method 486
SetPlannedTaskBarColor method 487
SetPopUpNodeData method 524
SetPopUpNodeRecord method 525
SetPosition method 977, 1000
SetProcessInstanceCriteria method 2623
SetProperties method
Grid class 1042
SetPropertyByName method
Component Interface class 631
Mobile class 1412
SetProperty method 2637
SetPSQueryPromptRecord method 2607
SetQueryString method 1288
setRangeFromFieldTranslates method
DataSourceParameter class 834
DataSourceSetting class 847
SetRBNodeData method 544
SetRBNodeRecord method 545
SetRecTerminator method 1001
SetReportIDCriteria method 2624
SetRule method
cube class 105
UserFunctionDefn class 124
SetRunControlData method
CONQRSMGR class 681
SetRuntimeDataXMLFile method 2608
SetRuntimeProperties method 2609
SetSecondFormat method 488
SetSelectedField method 185
SetSlice method 48
SetSMTPParam method
MCFOutboundEmail class 1186
SMTPSession 1207
SetSMTPParam property 1208
SetSQL method 9
SetStatus method 1289, 1331

SetSubstitution method 767
SetTableXScrollbar method 489
SetTaskAppData method 489
SetTaskAppDataTitles method 490
SetTaskBarURL method 491
SetTaskData method 492
SetTaskDependencyChildID method 492
SetTaskDependencyData method 493
SetTaskDependencyParentID method 494
SetTaskDependencyType method 494
SetTaskDependencyURL method 496
SetTaskDrill method 497
SetTaskExpanded method 497
SetTaskHints method 498
SetTaskID method 499
SetTaskLabel method 499
SetTaskLevel method 500
SetTaskMilestone method 501
SetTaskName method 502
SetTaskProgressBarColor method 503
SetTaskProgress method 503
SetTemplate method 10
setting
 drilling URLs 1896
 XML Publisher custom runtime parameters 2610
SettingsCompleted property
 DataSource class 830
Settings property
 DataSource class 830
SetTraceLevel method 1537
SetTrackingURL method 1944
SetupCompVarsAndRcpts method 1462
SetupGenericVars method 1463
SetUserIdCriteria method 2624
SetVariableBounds method 1574
SetVariableType method 1576
SetWBSNumbering method 504
SetXAxisLabels methods 452, 545
SetXmlDoc method 1290
SetYAxisLabels methods 452, 546
SetYearFormat method 505
Severity property 2484
SF_MAXMINUTES_ALLMSGS property
 Utility class 895
SF_MAXROWOPTION_ALLMSGS property
 Utility class 896
SF_MAXROWOPTION_LATESTMSG property
 Utility class 896
SF_PAGINGOPTION_NOPAGING property
 Utility class 897
SF_PAGINGOPTION_SEGMENTED property
 Utility class 897
Shift method 278
shortcut considerations
 Field class 936
 Grid class 1036
 Page class 1581
 Record class 2078
 Row class 2112
 Rowset class 2128
Shortname property 2057
ShortName property 1992
ShortTranslateValue property 968
ShowAllRows method 2159
ShowColumns method
 Grid class 1043
ShowCrossHair property 464
showException method
 Utility class 869
ShowFormattedXML property
 CONQRSMGR class 696
ShowGridLines property 167
ShowHierarchy method 49
showInvalidValueException method
 Utility class 869
ShowNodeDescription property 553
ShowRelatedField property
 DataSourceSetting class 852
ShowRequiredCue property 1438
ShowRequiredFieldCue property 969
ShowTaskLabels property 515
ShutDown method 1539
signon
 authenticating single signon 1066
 ByPassSignOn property 1067
 PeopleSoft sessions 2180
Simple Object Access Protocol (SOAP)
 See SOAP
Size property
 MCFInboundEmail class 1156
 Message class 1309
SkillLevel property 2440
SkipValidityChecks property
 DataSourceParameter class 839
SlicerVisible property 168
SmartNavigation
 GenABNMenuElement method 2305, 2329
 GenBreadCrumbs method 2330
 GenRelatedActions method 2331
 invalid characters 2296
 LoadABNChart method 2307, 2337
 LoadABNChartOrdered method 2308, 2338
 restrictions on trees 2296
SmartZero property 969
SMTPPort property 1200
SMTPServer property 1200
SMTPSession class
 constructor 1116
 list of methods, properties, and returns 2734
 methods
 See Also SMTPSession class methods
SMTPSession class methods
 Send 1205
 SendAll 1206
 SetSMTPParam 1207
SMTPSession class properties
 BackupServer 1208
 BackupSSLClientCertAlias 1209
 BackupSSLPort 1209
 BackupUserName 1209
 BackupUserPassword 1210
 BackupUseSSL 1210
 IsAuthenticationReqd 1210
 Port 1211
 Server 1211
 SetSMTPParam 1208
 SSLClientCertAlias 1211
 SSLPort 1212
 UsedDefaultConfig 1212
 UsedPrimaryServer 1212
 UserName 1212
 UserPassword 1213
 UseSSL 1213
SMTPSSLClientCertAlias property 1200
SMTPSSLPort property 1201

SMTPUserName property 1201
SMTPUserPassword property 1201
SMTPUseSSL property 1202
SOAP
  messages 2212
  SOAPDoc class *See Also* SOAPDoc class
  SOAPDoc objects
    See Also SOAPDoc objects
  understanding 2211
  validating documents 2229
SOAPDoc
  class *See Also* SOAPDoc class
  objects *See Also* SOAPDoc objects
SOAPDoc class
  creating documents 2235
  examples 2235
  list of functions, methods, properties, returns 2769
  methods *See Also* SOAPDoc class methods
  properties
    See Also SOAPDoc class properties
  reading documents 2236
  understanding 2211, 2214
  using SOAP XML text 2236
SOAPDoc class methods
  AddBody 2219
  AddEnvelope 2220
  AddFault 2222
  AddHeader 2224
  AddMethod 2225
  AddParm 2227
  GetParmName 2227
  GetParmValue 2228
  ValidateSOAPDoc 2217, 2229
SOAPDoc class properties
  BodyNode 2231
  EnvelopeNode 2231
  FaultCode 2231
  FaultCodeS 2232
  FaultString 2232
  HeaderNode 2233
  MethodName 2234
  MethodNode 2234
  ParmCount 2234
  understanding 2230
  XmlDoc 2235
SOAPDoc objects
  accessing sections 2216
  body section 2215
  creating 2217
  declaring/scope 2218
  envelope section 2215
  fault section 2217
  header section 2214
  method section 2216
  understanding 2214
  using 2218
  using XmlDoc/XmlNode methods/properties 2534
  validating 2217
SOAP messages 2212
Solve method 1577
solvers
  activating 1565
  deactivating 1567
sorting
  records 2136
  Rowset class Sort method 2160
Sort method
  Array class 279
  QueryDBRecordField collection 2051
  Rowset class 2160
SortSpecifications property 2481
SourceComponent property 1458
SourceMarket property 1458
SourceMenu property 1458
SourceMsgName property 2289
SourceMsgVersion property 2289
SourceNode property 1371, 2289
Source property 2196
SourceReportPath property 1832
spidering
  FSYS search indexes 2518
  HTTP search indexes 2514
  Search Start Options collection 2520
  using for mobile devices 2268
split2D method
  Utility class 871
split method
  Utility class 870
SQL
  binding/executing statements 2241
  case sensitivity 2240
  class *See Also* SQL class
  definitions *See Also* SQL definitions
  fetching from a select 2243
  objects *See Also* SQL objects
  opening/processing 2253
  query classes *See Also* query classes
  reusing a cursor 2244
  SetSQL method 9
  SQL property 1955
SQL class
  application engine considerations 2247
  global declaration considerations 2247
  list of functions, methods, properties, returns 2770
  methods *See Also* SQL class methods
  properties *See Also* SQL class properties
  ReuseCursor property 2244
  reusing a cursor 2244
  understanding 2239
  understanding SELECT styles 2243
  using "Array of Any" 2243
SQL class methods
  Close 2248
  Execute 2249
  Fetch 2250
  Open 2252
SQL class properties
  BulkMode 2254
  IsOpen 2255
  LTrim 2256
  ReuseCursor 2256
  RowsAffected 2257
  Status 2257
  TraceName 2258
  Value 2259
SQL definitions
  creating 2241
  understanding 2239
SQL objects
  declaring/scope 2247
  reusing cursors 2244
  understanding 2239
SQL property 1955

SQLStatement property 2209
SQLStatementVariables property 2209
SqlText property 969
SSBs property 2209
SSLClientCertAlias property 1211
SSLPort property 1212
StackTrace Exception class property 770
StartFromPageNum property 2630
StartNum property 2631
StartOpts property
    Search FS Options class 2520
    Search HTTP Options class 2518
Stat_Active property
    CONQRS_CONST class 704
Stat_InActive property
    CONQRS_CONST class 704
Stat_InProgress property
    CONQRS_CONST class 704
statements 764
State property 2354
states
    Business Interlink objects 310
    managing 1088
    returning to users 2183
status
    building status response messages 1507
    checking for optimization engines 1498, 1518
    ConflictStatus property 2277
    email 1108
    LockStatus property 2396
    MCFGetMail error codes 1114
    ProcessRequest class Status property 1877
    RunStatus property 1875
    sending for optimization 1501
    SQL class Status property 2257
    Tree class Status property 2400
    writing status messages to optimization engine logs 1567
StatusNotifyOptions property 1202
StatusNotifyReturn property 1203
Status property 2290, 2613
    CONQRSMGR class 697
    MCFGetMail class 1146
    MCFInboundEmail class 1156
    ProcessRequest class 1877
    SQL class 2257
    Tree class 2400
StopAtError property 372
StopOnFirstError property 635
StorageType property 1695
StoredFormat property 970
StoreEmail method 1160
stringToDate method
    Utility class 872
stringToDatetime method
    FeedDoc class 909
    Utility class 872
structured messages 1245
StructureName property 2401
Structure property 2401
Style property
    Chart class appearance 464
    Field class 971
    Gantt class appearance 516
    OrgChart class appearance 541, 553
    Row class 2124
styles, iScript 1056
StyleSheet property
    Chart class appearance 464
    Gantt class appearance 516
    TabDefinition class 1744
style sheets
    default classes for charts 384
    using for charts 383
stylesheets, iScript 1056
Subarray method 282
subclasses
    downcasting 219
    overriding superclass properties 214
    understanding 189, 192
    using instance variables 218
    using private methods 198
    using protected methods 194
Subject property 1131
    MCFOutboundEmail class 1203
    Notification class 1458
    NotificationTemplate class 1465
SubName property 1310
SubQueueName property 1310
subscription
    subscribing to partial records 1251
subscription PeopleCode 2180
SubscriptionProcessId property 1311
Substitute method 282
SubstitutionCount Exception class property 771
SubType property 1178
subtyping 192
SummaryLevelNumber property 2418
SummarySetId property 2417
SummaryText property
    AnalyticGrid class 168
    Grid class 1045
SummaryTreeName property 2418
SummaryUserKeyValue property 2418
superclasses
    referencing 218, 225
    understanding 192
SuppressPeopleCodeOnNode property 541
SuspendFormatting property 2190
SwitchAsyncEventUserContext method
    IntBroker class 1332
SwitchLevel method 2345
SybBindInfo property 2209
SybFetchInfo property 2210
synchronization
    considerations 2283
    events
        See Also synchronization server events
    filtering 2270
    implementing custom data distribution 2267
    Mobile Agent 1386
synchronization server events
    understanding 2261
    using 2264, 2265
Synchronizing property 2281
synchronous mode
    loading analytic instances 1491
    running optimization transactions 1492
    running transactions 1499
    understanding the RunSynch method 1536
syncid property 1480
SyncIDs property 2281
SyncOrder property 187
SyncRequest method
    IntBroker class 1332

message class 1291
SyncServer
class *See Also* SyncServer class
objects *See Also* SyncServer objects
SyncServer class
list of deprecated methods, properties, returns, variables 2880
methods *See Also* SyncServer class methods
properties
See Also SyncServer class properties
understanding 2261, 2264, 2271
SyncServer class methods
AddReference 2271
AddReferenceType 2272
CheckForChanges 2273
Notify 2275
SelectAll 2276
SyncServer class properties
ClientPlatform 2277
ConflictAlgorithm 2277
ConflictStatus 2277
ExcludeProperty 2278
ExportLocation 2278
IsReferenceUnresolved 2279
LastSyncDateTime 2279
LastSyncRowCount 2279
PropertyValue 2280
SessionID 2280
Synchronizing 2281
SyncIDs 2281
TypeID 2282
validateID 2282
validateRowCount 2282
validateVersion 2283
SyncServer objects
accessing/declaring/scope 2271
understanding 2263, 2271
SyncServiceTimeout property 1371
SysCon class 1091
SystemIDFieldName property 2108
system variables
%Request 1059
%Response 1059
%Super 218
%SyncServer 2263, 2271
%This 210, 211
%ThisMobileObject 1389
iScript classes 2721
mapping deprecated to new 2869
supported in mobile PeopleCode 1442
SysVar Java class 1091
SYSVAR_INVALID property
Utility class 898
SysVar class 1091

T

TabDefinition
class *See Also* TabDefinition class
collection *See Also* TabDefinition collection
TabDefinitions property 1642
TabDefinition class
list of methods, properties, returns 2825
methods 1737
properties
See Also TabDefinition class properties
understanding 1737
TabDefinition class properties
AssignedPagelets 1738
Attributes 1739
Author 1739
AuthorAccess 1739
AvailableCategories 1738
AvailablePagelets 1739
ColumnLayout 1740
CreationDate 1740
Description 1740
DynamicCategories 1740
HelpID 1741
HtmlText 1741
IsHideActionBar 1741
IsLayoutLocked 1741
IsRenameable 1742
Label 1742
LayoutBehavior 1742
Name 1743
OwnerId 1743
Product 1743
PublicAccess 1743
QualifiedURL 1744
SequenceNumber 1744
StyleSheet 1744
ValidFrom 1744
ValidTo 1745
TabDefinition collection
list of methods, properties, returns 2826
methods
See Also TabDefinition collection methods
properties 1748
understanding 1745
TabDefinition collection methods
DeleteItem 1745
First 1746
InsertItem 1747
ItemByName 1747
Next 1748
TabDefinition collection properties 1748
TabDefinitions property 1642
tables
flat table methods 305
INSTALLATION 973
OPTIONS 973
PS_MESSAGE_LOG 1496
TabName property 1792
target content references 1604
Task class
list of methods, properties, returns 2776
methods *See Also* Task class methods
properties *See Also* Task class properties
understanding 2453
Task class methods
Close 2453
Enqueue 2454
Refresh 2456
RefreshStatus 2456
Task class properties
AgentID 2457
ApplicationData 2457
Comments 2458
EnqueueTime 2458
EscalationTime 2458
Language 2458
OriginalTime 2458
OverFlowTime 2459

PhysicalQueueID 2459
TiedAgentID 2459
Task constructor 2430
TaskDependencyLineType property 516
TaskList class
list of methods, properties, returns 2777
methods 2459
properties *See Also* TaskList class properties
understanding 2459
TaskList class properties
AgentID 2460
PhysicalQueueID 2460
Task 2461
TaskType 2461
Total 2461
TaskList constructor 2431
TaskMilestoneGlyph property 517
Task property 2461
tasks
Task class *See Also* Task class
TaskList class *See Also* TaskList class
TaskList class Task property 2461
TaskTitle property 517
TaskType property 2461
template content references 1604
TemplateId property 1466
TemplateObject property
Content Reference class 1696
ContentReference links 1732
NodeTemplate class 1659
PortalRegistry class 1643
Template property
Content Reference class 1696
ContentReference links 1731
Notification class 1459
templates
content references 1604
deleting 1609
editing OPT_CALL 1506
editing PT_OPTCALL 1502
NodeTemplate class
See Also NodeTemplate class
NotificationTemplate class
See Also NotificationTemplate class
SetTemplate method 10
TemplateObject property 1696
TemplateObject property (ContentReference links) *See Also* TemplateObject property
Template property 1696
Template property (ContentReference links)
See Also Template property
TemplateType property 1697
TemplateType property (ContentReference links) *See Also* TemplateType property
TemplateType property
Content Reference class 1606, 1697
ContentReference links 1732
NotificationTemplate class 1466
TerminateLines property 1012
Terminate method 24
TextAngle property 2641
TextFontName property 2641
TextFontSize property 2641
Text property
MCFBodyPart class 1127
MCFEmail class 1131
MCFOutboundEmail class 1203
NotificationTemplate class 1466
PSMessages class 2197
QueryExpression class 2021
Watermark class 2641
TextStartPosX property 2642
TextStartPosY property 2642
third-party integrators 234
TiedAgentID property 2459
timedout property 1480
Timeout property 1074
timeouts
Component Interface 607
loading analytic instances 1491
running optimization transactions 1493
Timeout property 1074
TimeStampFieldName property 2109
TimeToWaitForResult property 1203
TimeZone property 1877
Title property
Feed class 797
FeedDoc class 917
FeedEntry class 929
ToolsRelease property 1649
TopRowNumber property 2169
ToString method 768
TotalAgents property 2453
TotalMemberName property 95
TotalPhysicalQueues property 2439
Total property 2461
TraceFile property 2210
TraceName property 2258
Trace Setting class properties
API 2204
COBOLStmtTimings 2204
ConnDisRollbackCommit 2204
DBSpecificCalls 2204
ManagerInfo 2205
NetworkServices 2205
NonSSBs 2205
OutputUNICODE 2205
PCExtFcnCalls 2206
PCFcnReturnValues 2206
PCFetchedValues 2206
PCIntFcnCalls 2206
PCListProgram 2207
PCParameterValues 2207
PCProgramStatements 2207
PCStack 2207
PCStartOfPrograms 2208
PCTraceProgram 2208
PCVariableAssignments 2208
RowFetch 2208
SQLStatement 2209
SQLStatementVariables 2209
SSBs 2209
SybBindInfo 2209
SybFetchInfo 2210
TraceFile 2210
Trace Settings
class *See Also* Trace Setting class properties
objects 2182
TraceSettings class 2791
TraceSettings objects 2182
TraceSettings property 2190
tracing
getting for request messages 1500
mobile PeopleCode 1394, 1396
setting for request messages 1499
setting options *See Also* Trace Settings

TraceName property 2258
transactionid property
WorklistEntry class 1480
TransactionId property 1312
TransactionID property 1372
WSWorklistEntry class 1484
transfer functions 763
transfers 611
TransformData class
about 2287
creating an object 2287
list of variables, properties, and returns 2772
properties
See Also TransformData class properties
understanding 2287
TransformData class properties
DestMsgName 2288
DestMsgVersion 2288
DestNode 2288
RejectTransform 2288
RoutingDefnName 2289
SourceMsgName 2289
SourceMsgVersion 2289
SourceNode 2289
Status 2290
XmlDoc 2290
Translatable property 1706
TranslateEffDtLogic property 1992
TranslateExpression property 1993
TranslateField property 1993
TranslateOption property 1993
translation 383
TreeBranchName property
Leaf class 2319
Level class 2325
Node class 2354
Tree class
list of methods, properties, returns 2863
methods *See Also* Tree class methods
properties *See Also* Tree class properties
understanding 2356
tree classes
handling errors 2294
implementing 2297
instantiating 2300
Leaf *See Also* Leaf class
Level *See Also* Level class
list of methods, properties, returns 2857
Node *See Also* Node class
relationship between 2292
Tree *See Also* Tree class
Tree Structure *See Also* Tree Structure class
understanding 2291
verifying leaf/node insertions 2295
Tree class methods
Audit 2356
AuditByName 2357
Close 2358
Copy 2359
Create 2360
Delete 2362
Exists 2363
FindLeaf 2364
FindNode 2365
FindRoot 2366
InsertRoot 2367
LeafExists 2368
LockTree 2369
NodeExists 2375
Open 2369
OpenAsOfDate 2371
OpenForExport 2373
OpenWholeTree 2374
Rename 2376
Save 2377
SaveAs 2378
SaveAsDraft 2379
SaveDraft 2381
SetImportMode 2382
TreeLocksNumber 2382
UnlockTree 2383
UpdateLock 2384
Tree class properties
AllValues 2385
AuditDetails 2386
Branches 2386
BranchImageName 2386
BranchLevel 2386
BranchName 2387
Category 2387
Description 2387
DuplicateLeaves 2388
EffDt 2388
HasDetailRanges 2389
HasLockedBranches 2389
IsBranched 2389
IsChanged 2390
IsOpen 2390
IsQueryTree 2390
IsValid 2391
IsVersionChanged 2391
IsWholeTree 2391
KeyBranchName 2392
KeyEffDt 2392
KeyName 2392
KeySetId 2393
KeyUserKeyValue 2393
LeafCount 2393
LeafImageName 2393
LeafOnClipboard 2394
LevelCount 2394
Levels 2394
LevelUse 2395
LockOwner 2395
LockStatus 2396
Name 2396
NodeColImageName 2397
NodeCount 2397
NodeExpImageName 2397
NodeOnClipboard 2397
ParentLevel 2398
ParentName 2398
PerformanceMethod 2398
PerformanceSelector 2399
PerformanceSelectorOption 2399
SetID 2400
Status 2400
Structure 2401
StructureName 2401
TreeImageName 2402
UserKeyValue 2402
UseUpdateReservation 2402
Tree collections
Branch *See Also* Branch collection
Level *See Also* Level collection
understanding 2294

TreeEffDt property
  Leaf class 2319
  Level class 2325
  Node class 2354
TreeEffectiveDate property
  Folder class 1673
TreeImageName property 2402
TreeLocksNumber method 2382
Tree Manager 2291
TreeName property
  Folder class 1674
  Leaf class 2319
  Level class 2325
  Node class 2355
tree objects
  declaring/scope 2297
  getting/opening 2292
  understanding 2291, 2356
trees
  analytic calculation engine class 14
  AnalyticModel class
    attaching 27
    accessing 33
  classes *See Also* tree classes
  collections *See Also* Tree collections
  creating 2297
  detaching from dimensions 29
  objects *See Also* tree objects
  structures *See Also* tree structures
  traversing hierarchies 2419
TreeSetId property
  Folder class 1675
  Leaf class 2319
  Level class 2325
  Node class 2355
Tree Structure class
  list of methods, properties, returns 2867
  methods
    See Also Tree Structure class methods
  properties
    See Also Tree Structure class properties
  understanding 2402
Tree Structure class methods
  Close 2403
  Copy 2403
  Create 2404
  Delete 2405
  Open 2405
  Rename 2406
  Save 2406
Tree Structure class properties
  Description 2407
  DetailComponent 2407
  DetailField 2408
  DetailMenu 2408
  DetailMenuBar 2409
  DetailMenuItem 2409
  DetailMultiNavigate 2409
  DetailPage 2410
  DetailRecord 2410
  IndirectionMethod 2411
  KeyName 2412
  LevelComponent 2412
  LevelMenu 2412
  LevelMenuBar 2413
  LevelMenuItem 2413
  LevelPage 2413
  Name 2414
  NodeComponent 2414
  NodeField 2415
  NodeMenu 2415
  NodeMenuBar 2415
  NodeMenuItem 2416
  NodeMultiNavigate 2416
  NodePage 2416
  NodeRecord 2417
  NodeUserKeyField 2417
  SummaryLevelNumber 2418
  SummarySetId 2417
  SummaryTreeName 2418
  SummaryUserKeyValue 2418
  Type 2419
TreeStructureName property
  Folder class 1676
tree structure objects 2402
tree structures
  class *See Also* Tree Structure class
  objects 2402
  portal registry *See Also* portal registry
  relationships between tree classes 2292
  understanding 2291
TreeUserKeyValue property
  Folder class 1677
  Leaf class 2320
  Level class 2325
  Node class 2355
try-catch statements 764
type checking 192
TypeID property 2282
Type property
  AnalyticTypeModelDefn class 184
  Chart class appearance 465
  comparison class 136
  CompIntfPropInfoCollection objects 653
  constant class 138
  Field class 971
  FunctionCall class 149
  MethodInfo 252
  Node class 2355
  Operation class 152
  PropertyInfo 255
  Query class 1955
  QueryDBRecordField class 2057
  QueryExpression class 2022
  QueryField class 1994
  SearchIndex class 2497
  Tree Structure class 2419
  variable class 154
typographic conventions cxvii

U

UID property 1156
Unbranch method 2345
Uniform Resource Identifier (URI) 1053
Uniform Resource Locator (URL) *See* URL
UniquePromptName property 2066
universal queue classes
  Agent class *See Also* Agent class
  AgentPhysQueueProps class
    See Also AgentPhysQueueProps class, AgentPhysQueueTasks class
  Broadcast class 2443
  constructors

See Also universal queue classes constructors
importing 2423
list of functions constructors, methods, properties, returns 2773
LogicalQueue class
See Also LogicalQueue class
MCFFactory class
See Also MCFFactory class
PhysicalQueue class
See Also PhysicalQueue class
scope 2422
Task class *See Also* Task class
TaskList class *See Also* TaskList class
understanding 2421
Util class *See Also* Util class
Universal Queue Classes
example 2463
universal queue classes constructors
Agent 2425
AgentPhysQueueProps 2426
AgentPhysQueueTasks 2426
Broadcast 2427
LogicalQueue 2428
MCFFactory 2428
PhysicalQueue 2429
Task 2430
TaskList 2431
Util 2433
universal queue objects 2424
universal queues
classes *See Also* universal queue classes
objects 2424
Unload method 24
UnlockTree method 2383
unpublishFromSites method
Feed class 788
UnsetDimFilter 50
UnsetDimSort method 51
UnsetSelectedField method 185
Unshift method 284
unstructured messages
understanding 1245
using XmlDoc messages 2218
UpdateConnectorResponseProperties method 297
UpdateData method 726
Updated property
FeedDoc class 917
FeedEntry class 929
UpdateFromRowset method
Document class 737
UpdateLock method 2384
Update method
IntBroker class 1334
Message class 1293
Record class 2102
WorklistEntry class 1475
Update mode 977
UpdateRanges method 2314
UpdateRunStatus method 1865
UpdateSegment method 1294
UpdateXmlDoc method 1335
updateXMPPServerUserData method
MCFIMSingleButton class 1243
URI 1053
URL
assembling to view iScripts 1056
Favorite class QualifiedURL property 1803
Favorite class URL property 1803
generating for portal content 1057
Pagelet class QualifiedURL property 1783
Pagelet class URL property 1784
Pagelet class URLType property 1784
understanding 1053
url property 1481
URL property 1697
URL property (ContentReference links)
See Also URL property
URLType property 1606, 1698
URLType property (ContentReference links)
See Also URLType property
UserTab class QualifiedURL property 1792
URL property
Content Reference class 1697
ContentReference links 1733
Favorite class 1803
Pagelet class 1784
using 1607
WorklistEntry class 1481
URLType property
Content Reference class 1606, 1698
ContentReference links 1733
Pagelet class 1784
Usage property 256
USAGETYPE_ADMINSPECIFIED property
Utility class 898
USAGETYPE_FIXED property
Utility class 898
USAGETYPE_INTERNAL property
Utility class 898
USAGETYPE_NOTUSED property
Utility class 898
USAGETYPE_SYSVAR property
Utility class 899
USAGETYPE_USERSPECIFIED property
Utility class 899
UsageType property
Content Reference class 1604, 1699
ContentReference links 1733
DataSourceParameter class 840
UseBurstValueAsOutputFileName property 2613
UseCount property 2066
UsedDefaultConfig property
MCFOutboundEmail class 1204
SMTPSession class 1212
usedefaultoutdestination property 2611
UsedPrimaryServer property
MCFOutboundEmail class 1204
SMTPSession class 1212
UseLocalTime property 2202
user-sorted grids 2141
UserFunctionDefn class methods 123
UserFunctionDefn class properties 124
user functions
accessing 82
names 82
rules 123
adding to analytic model 62
aggregation 94
class properties 124
commenting 124
copying 67
deleting 73
naming 125
renaming 88
setting

rules 124
UserHomepage
class *See Also* UserHomepage class
UserHomepage class
list of methods, properties, returns 2834
methods 1789
properties 1790
understanding 1789
UserHomepage class methods 1789
UserHomepage class properties 1790
UserId property 1790
UserID property 1146
UserKeyValue property 2402
UserName property 1212, 1372
UserPassword property 1213
UserPersonalizationPage property
DataSource class 830
User property 1156
UsersPart method 128
UserTab 1791
class *See Also* UserTab class
collection *See Also* UserTab collection
property 1790
UserTab class
list of properties, returns 2834
properties *See Also* UserTab class properties
understanding 1791
UserTab class properties
ColumnLayout 1791
Label 1791
QualifiedURL 1792
SelectedPagelets 1792
SequenceNumber 1792
TabName 1792
UserTab collection
list of methods, properties, returns 2834
methods
See Also UserTab collection methods
properties 1796
understanding 1793
UserTab collection methods
DeleteItem 1793
First 1794
InsertItem 1794
ItemByName 1795
Next 1795
UserTab property 1790
UserValues property
DataSourceParameter class 840
UsesCube method 120
UsesDimension method
CubeCollectionDefn class 121
CubeDefn class 105
UseSimpleURL method 1080
UseSpaceForNull property 1013
UseSSL property 1213
UseUpdateReservation property 2402
Util class 2462
Util constructor 2433
Utility class
about 853
import statements 854
list of methods 2702
list of properties 2704
UTILITY class
about 710
import statements 710
list of methods 2686
utility classes 224
Utility class methods
constructor 854
dateStringToUserPref 854
datetimeToString 855
dateToString 855
decodeXML 856
encodeXML 857
evaluateSysVar 858
genNameSpaceID 859
getExceptionText 859
getFeedDoc 860
getFeedMimeType 861
getFieldTranslates 861
getNodeValue 862
getUserDateFormat 863
getUserDatetimeFormat 863
getUserInfo 864
httpStringToDatetime 864
join 865
join2D 866
setMessageHeadersAndMimeType 867
setNodeValue 868
showException 869
showInvalidValueException 869
split 870
split2D 871
stringToDate 872
stringToDatetime 872
Utility 854
validateSysVar 873
viewStringAsAttachment 874
UTILITY class methods
CheckQryForTreePrompt 710
CheckQrySecurity 711
constructor 710
GetQueryScopeByName 712
UTILITY 710
ValidateObjectID 713
Utility class properties
ATTACHMENT_URL 875
AUTHTYPE_PERM 875
AUTHTYPE_ROLE 875
DSPARAMETER_INCREMENTAL 875
DSPARAMETER_MAXROW 876
DSPARAMETER_SF_MAXMINUTES 876
DSPARAMETER_SF_PAGING 876
EDITTYPE_NOTABLEEDIT 877
EDITTYPE_PROMPTTABLE 877
EDITTYPE_TRANSLATETABLE 877
EDITTYPE_YESNO 877
FEEDATTRIBUTE_AUTHOR 878
FEEDATTRIBUTE_CLOUD 878
FEEDATTRIBUTE_COMPLETE 878
FEEDATTRIBUTE_CONTRIBUTOR 878
FEEDATTRIBUTE_COPYRIGHT 879
FEEDATTRIBUTE_EXPIRES 879
FEEDATTRIBUTE_ICONURL 879
FEEDATTRIBUTE_LOGOURL 879
FEEDATTRIBUTE_MANAGINGEDITOR 880
FEEDATTRIBUTE_MAXAGE 880
FEEDATTRIBUTE_PERSINSTRUCTION 880
FEEDATTRIBUTE_RATING 880
FEEDATTRIBUTE_SKIPDAYS 880
FEEDATTRIBUTE_SKIPHOURS 881
FEEDATTRIBUTE_TEXTINPUT 881

FEEDATTRIBUTE_TTL 881
FEEDATTRIBUTE_WEBMASTER 881
FEEDATTRIBUTE_XSL 882
FEEDAUTHTYPE_ALL 882
FEEDAUTHTYPE_ANONYMOUS 882
FEEDAUTHTYPE_DEFAULT 882
FEEDCACHETYPE_NONE 882
FEEDCACHETYPE_PRIVATE 883
FEEDCACHETYPE_PUBLIC 883
FEEDCACHETYPE_ROLE 883
FEEDFORMAT_ATOM10 883
FEEDSECUTYPE_PUBLIC 883
FEEDSECUTYPE_REALTIME 884
FEEDSECUTYPE_SELECTED 884
FEEDTYPE_DYNAMIC 884
FEEDTYPE_PREPUBLISHED 884
FIELDTYPE_CHARACTER 884
FIELDTYPE_DATE 885
FIELDTYPE_DATETIME 885
FIELDTYPE_LONGCHARACTER 885
FIELDTYPE_NUMBER 885
FIELDTYPE_SIGNEDNUMBER 885
FIELDTYPE_TIME 886
IBSOTYPE_ASYNC 886
IBSOTYPE_SYNC 886
IBSOTYPE_UNKNOWN 886
ICONURL_FEED_A 886
ICONURL_FEED_IA 887
INCREMENTALOPTION_NO 887
INCREMENTALOPTION_YES 887
LINKTYPE_FIRST 888
LINKTYPE_LAST 888
LINKTYPE_NEXT 888
LINKTYPE_PREVIOUS 889
MIMETYPE_ATOM 889
MIMETYPE_OPML 889
MIMETYPE_XML 890
OPERATINGMODE_AUTHORIZATION 890
OPERATINGMODE_DEFAULT 890
OPERATINGMODE_DELETION 890
OPERATINGMODE_EXECUTION 891
OPERATINGMODE_EXECUTION_NOENTRY 891
QUERYPARAMETER_CHILDFEEDID 891
QUERYPARAMETER_DATATYPEID 891
QUERYPARAMETER_DEFLOCALNODE 892
QUERYPARAMETER_DSSCOUNT 892
QUERYPARAMETER_DSSNAME 892
QUERYPARAMETER_DSSVALUE 892
QUERYPARAMETER_FEEDFORMAT 892
QUERYPARAMETER_FEEDID 893
QUERYPARAMETER_FEEDLIST 893
QUERYPARAMETER_FEEDTYPE 893
QUERYPARAMETER_IBTRANSID 893
QUERYPARAMETER_IFMODIFIEDSINCE 893
QUERYPARAMETER_IFNONEMATCH 894
QUERYPARAMETER_KEYWORD 894
QUERYPARAMETER_LANGUAGE 894
QUERYPARAMETER_NODENAME 894
QUERYPARAMETER_PAGENUM 894
QUERYPARAMETER_PORTALNAME 895
QUERYPARAMETER_PTPPB_SEARCH_MODE 895
QUERYPARAMETER_PTPPB_SEARCH_TEXT 895
RequestInfo 895
SF_MAXMINUTES_ALLMSGS 895
SF_MAXROWOPTION_ALLMSGS 896
SF_MAXROWOPTION_LATESTMSG 896
SF_PAGINGOPTION_NOPAGING 897
SF_PAGINGOPTION_SEGMENTED 897
SYSVAR_INVALID 898
USAGETYPE_ADMINSPECIFIED 898
USAGETYPE_FIXED 898
USAGETYPE_INTERNAL 898
USAGETYPE_NOTUSED 898
USAGETYPE_SYSVAR 899
USAGETYPE_USERSPECIFIED 899
XMLCHILDELEMENTS_CLOUD 899
XMLCHILDELEMENTS_PERSON 899
XMLCHILDELEMENTS_TEXTINPUT 900
XMLELEMENT_DAY 900
XMLELEMENT_DESCRIPTION 900
XMLELEMENT_DOMAIN 900
XMLELEMENT_EMAIL 901
XMLELEMENT_HOUR 901
XMLELEMENT_LINK 901
XMLELEMENT_NAME 901
XMLELEMENT_PATH 901
XMLELEMENT_PORT 902
XMLELEMENT_PROTOCOL 902
XMLELEMENT_REGISTERPROCEDURE 902
XMLELEMENT_TITLE 902
Utility method
Utility class 854
UTILITY method
UTILITY class 710
Utility property
DataSource class 831
DataSourceParameter class 840
DataSourceSetting class 852
Feed class 798
FeedFactory class 814

V

ValidateAddress method 1171
ValidateData method
Document class 737
ValidateEnum method 1413
validateID property 2282
ValidateIfFieldsMapped method
CONQRSMGR class 683
Validate method 89
CONQRSMGR class 682
ValidateObjectID method
UTILITY class 713
validateRowCount property 2282
ValidateRunControlParms method
CONQRSMGR class 683
ValidateSOAPDoc method 2217, 2229
validateSysVar method
Utility class 873
validateValue method
DataSourceParameter class 835
validateVersion property 2283

ValidFrom property
Content Reference class 1699
ContentReference links 1734
Folder class 1678
TabDefinition class 1744
using 1595
ValidSentAddresses property 1204
ValidTo property
Content Reference class 1700
ContentReference links 1734
Folder class 1678
TabDefinition class 1745
using 1595
ValidUnsentAddresses property 1205
ValueDimensionName property 108
ValueForDisplay property
DataSourceParameter class 841
Value property
AttributeValue class 1706
constant class 138
Cookie class 1084
DataSourceParameter class 841
DataSourceParameterValue class 845
DataSourceSetting class 853
Field class 972
Primitive class 747
QueryListValue class 2026
Query Metadata class 2036
SearchField class 2491
Search Start Options class 2525
SQL class 2259
using character values 973
using INSTALLATION/OPTIONS tables 973
ValuesDifferFromLastSync method 1414
variable class
creating 153
method 153
Name property 154
properties 154
Type property 154
Variable property 133
variables
%Synch and %Asynch 1509
array variables 258
declaring private instance
See Also instance variables, instance variables
fetching data from a select 2243
JavaObject 1088
overriding 218
PCVariableAssignments property 2208
placing declarations in application classes 217
SetVariableBounds method 1574
SetVariableType method 1576
SQLStatementVariables property 2209
supporting for mobile PeopleCode 1397
system *See Also* system variables
typographic conventions cxviii
using component variables 606
using in application classes
217, *See Also* global variables
using local with PortalRegistry
See Also local variables
VeggieKey property 2499
Verified property 727
Verity search classes
list of methods, properties, returns 2800
understanding 2465
Version property 1312
DocumentKey class 740
VerticalSpace property 542
ViewAllStyle property 553
viewStringAsAttachment method
Utility class 874
Visible property
DataSourceSetting class 853
Field class 973
GridColumn class 1048
ListViewAttrs class 1434
Page class 1585
PeerDefaultAttributes class 1441
PropertyAttrs class 1438
Row class 2125
VisitedNodes property 1372
Visual Basic
accessing the API Repository 239
handling errors 2181
inserting data in component interfaces 664
referencing PortalRegistry objects 1820

W

WarningCount property 2484
WarningPending property 2190
WasSaved method 1415
Watermark class
about 2638
import statements 2639
Watermark class methods
constructor 2639
Watermark 2639
Watermark class properties 2641
ImageFile 2639
ImageFileLowerLeftX 2640
ImageFileLowerLeftY 2640
ImageFileUpperRightX 2640
ImageFileUpperRightY 2640
PageIndex 2641
Text 2641
TextAngle 2641
TextFontName 2641
TextStartPosX 2642
TextStartPosY 2642
Watermark method
Watermark class 2639
Watermark property 2634
web
setting attributes 1844
Web
creating libraries 1053
web services
WSWorklistEntry class 1482
Width property 466, 518, 542, 554
WinMessage function 1496
wl_priority property 1481
wldaystoselect property 1481
wldaystowork property 1481
Workflow Notification classes
See Notification classes
Worklist class
list of methods, properties, returns 2750
methods 1467

understanding 1466
Worklist constructor 1451
WorklistEntry
creating 1484
updating 1486
WorklistEntry class
list of methods, properties, returns 2750
methods
See Also WorklistEntry class methods
properties
See Also WorklistEntry class properties
understanding 1467
WorklistEntry class methods
Create 1468
GetResponseStatus 1469
Reassign 1470
Save 1471
SaveWithCustomData 1472
SelectByKey 1473
SelectByMessageId 1474
Update 1475
WorklistEntry class properties
actiondttm 1476
busactivity 1476
buseventname 1476
busprocname 1476
commentshort 1477
do_replicate_flag 1477
instanceid 1477
instselecteddttm 1477
inststatus 1478
insttimeoutdttm 1478
instworkeddttm 1478
IsCreatedViaWebService 1478
oprid 1479
originatorid 1479
prevoprid 1479
requestmessageid 1479
ResponseStatus 1479
syncid 1480
timedout 1480
transactionid 1480
url 1481
wl_priority 1481
wldaystoselect 1481
wldaystowork 1481
worklistname 1482
WorklistEntry constructor 1451
worklistname property 1482
WorkLoad property 2441
work records 383
World Wide Web Consortium (W3C) 2211
Writeable property 187
WriteBlobToFile method 1415
WriteLine method
File class 1002
Response class 1081
Write method 1081
WriteRaw method 1003
WriteRecord method
File class 1004
File Layout example 1017
WriteRowset method
File class 1006
File Layout example 1021
WriteString method 1008
WSA_Action property 1373
WSA_FaultTo property 1373
WSA_MessageID property 1373
WSA_ReplyTo property 1373
WSA_To property 1374
WSWorklistEntry class
list of methods 2752
list of properties 2752
properties 1483
understanding 1482
WSWorklistEntry classes
worklist entries
See Also WSWorklistEntry class
WSWorklistEntry class methods
OnError 1482
OnNotify 1483
WSWorklistEntry class properties
InstanceID 1484
TransactionID 1484
WSWorklistEntry constructor 1452

X

X-Priority 1105
XAxisBoxNum property 554
XAxisCrossPoint property 467
XAxisCross property 466
XAxisLabelOrient property 468
XAxisLabelStyle property 554
XAxisMax property 468
XAxisMin 468
XAxisPosition property 518
XAxisPrecision property 469
XAxisScaleResolution property 469
XAxisStyle property 470
XAxisTicks property 470
XAxisTitleOrient property 471
XAxisTitle property 470, 555
XAxisTitleStyle property 471, 555
Xlat property 653
XML
building responses 309
defining namespaces 2531
parsing requests 309
SOAP *See Also* SOAP
using with file definitions 1004, 1007
using with ReadRowset method 995
XMLCHILDELEMENTS_CLOUD property
Utility class 899
XMLCHILDELEMENTS_PERSON property
Utility class 899
XMLCHILDELEMENTS_TEXTINPUT property
Utility class 900
XMLDataFileName property
CONQRSMGR class 697
XMLDataFullName property
CONQRSMGR class 697
XmlDoc
objects *See Also* XmlDoc objects
property 2235
XmlDoc class
SOAPDoc considerations 2214
using 2530
XmlDoc classes 2529
list of functions, methods, properties, returns 2778
XmlDoc class methods 2536
XmlDoc class properties 2552

XmlNode class methods 2553
XmlNode class properties 2589
XmlDoc class methods
CopyRowset 2536
CopyToPSFTMessage 2537
CopyToRowset 2539
CreateDocumentElement 2541
CreateDocumentType 2542
GenFormattedXmlString 2544
GenXmlFile 2545
GenXmlString 2546
GetElementsByTagName 2546
LoadIBContent 2548
ParseXmlFromURL 2549
ParseXmlString 2551
XmlDoc objects
creating 2531
declaring/scope 2535
using 2530
XmlDoc property 2235, 2290
XMLELEMENT_DAY property
Utility class 900
XMLELEMENT_DESCRIPTION property
Utility class 900
XMLELEMENT_DOMAIN property
Utility class 900
XMLELEMENT_EMAIL property
Utility class 901
XMLELEMENT_HOUR property
Utility class 901
XMLELEMENT_LINK property
Utility class 901
XMLELEMENT_NAME property
Utility class 901
XMLELEMENT_PATH property
Utility class 901
XMLELEMENT_PORT property
Utility class 902
XMLELEMENT_PROTOCOL property
Utility class 902
XMLELEMENT_REGISTERPROCEDURE property
Utility class 902
XMLELEMENT_TITLE property
Utility class 902
XmlNode class
about 2553
handling errors 2534
list of methods, properties, returns 2779
understanding 2533
XmlNode class methods
AddAttribute 2553
AddAttributeNS 2554
AddCDataSection 2555
AddComment 2557
AddElement 2558
AddElementNS 2559
AddEntityReference 2560
AddNode 2561
AddProcessInstruction 2562
AddText 2563
CopyNode 2564
FindNode 2565
FindNodes 2566
GenXmlString 2567
GetAttributeName 2568
GetAttributeValue 2569
GetCDataValue 2569
GetCDataValues 2570
GetChildNode 2571
GetElement 2572
GetElements 2573
GetElementsByTagName 2574
GetElementsByTagNameNS 2575
InsertCDataSection 2576
InsertComment 2577
InsertElement 2579
InsertElementNS 2580
InsertEntityReference 2581
InsertNode 2582
InsertProcessInstruction 2584
InsertText 2585
RemoveAllChildNode 2586
RemoveChildNode 2587
XmlNode class properties
AttributesCount 2589
ChildNodeCount 2589
Index 2590
IsNull 2590
LocalName 2590
NamespaceURI 2591
NextSibling 2591
NodeName 2591
NodePath 2591
NodeType 2592
NodeValue 2592
ParentNode 2593
Prefix 2593
PreviousSibling 2593
XmlNode objects
accessing/traversing 2534
declaring/scope 2535
XML Publisher
engine classes 2596, 2627
flow diagram 2597
report manager definition classes 2596, 2600
search classes 2596, 2613
setting custom runtime parameters 2610
understanding 2595
XML Publisher classes 2595
data types 2598
error handling 2597
example 2642
generating reports 2642
importing 2598
instantiating objects 2598
list of constructors, methods, properties, returns 2782
list of deprecated methods and properties 2886
list of methods and properties not supported 2887
object scope 2598
PageNumber class, about 2627
PageNumber class constructor 2627
PageNumber class import statements 2627
PageNumber class properties 2628
PDFMerger class, about 2631
PDFMerger class constructor 2631
PDFMerger class import statements 2631
PDFMerger class method 2632
PDFMerger class properties 2633
Properties class, about 2635
Properties class constructor 2635
Properties class import statements 2635
Properties class methods 2636

publishing reports 2642
Report class, about 2613
Report class constructor 2614
Report class import statements 2614
Report class properties 2615
ReportDefn class, about 2600
ReportDefn class constructor 2600
ReportDefn class import statements 2600
ReportDefn class methods 2601
ReportDefn class properties 2611
ReportManager class, about 2618
ReportManager class constructor 2618
ReportManager class import statements 2618
ReportManager class methods 2619
scope 2598
SearchAttribute class, about 2625
SearchAttribute class constructor 2625
SearchAttribute class import statements 2625
SearchAttribute class properties 2626
search operator values 2599
terminology 2596
understanding 2595
Watermark class, about 2638
Watermark class constructor 2639
Watermark class import statements 2639
Watermark class properties 2639
XRotationAngle property 471

Y

Yahoo instant messaging 1233
YAxisCrossPoint property 472
YAxisLabelOrient property 472
YAxisLabelStyle property 556
YAxisMax property 473
YAxisMin property 473
YAxisPosition property 518
YAxisPrecision property 473
YAxisScaleResolution property 474
YAxisStyle property 474
YAxisTicks property 475
YAxisTitleOrient property 475
YAxisTitle property 475, 556
YAxisTitleStyle property 476, 556
YesNo property 654
YRotationAngle 476

Z

ZeroExtend property 1016
ZoneOptions property 2500
ZRotationAngle 476

