
PeopleSoft Enterprise Advanced Configurator 9 PeopleBook

August 2006

PeopleSoft Enterprise Advanced Configurator 9 PeopleBook
SKU CRM9CFG-B 0806
Copyright © 2001 - 2006, Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are “commercial computer software” or “commercial technical data” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee’s responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Open Source Disclosure

Oracle takes no responsibility for its use or distribution of any open source or shareware software or documentation and disclaims any and all liability or damages resulting from use of said software or documentation. The following open source software may be used in Oracle’s PeopleSoft products and the following disclaimers are provided.

Apache Software Foundation

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>). Copyright © 2000-2003. The Apache Software Foundation. All rights reserved. Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

OpenSSL

Copyright © 1998-2005 The OpenSSL Project. All rights reserved.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT “AS IS” AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Loki Library

Copyright © 2001 by Andrei Alexandrescu. This code accompanies the book: Alexandrescu, Andrei. “Modern C++ Design: Generic Programming and Design Patterns Applied”. Copyright © 2001 Addison-Wesley. Permission to use, copy, modify, distribute and sell this software for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

Helma Project

Copyright © 1999-2004 Helma Project. All rights reserved. THIS SOFTWARE IS PROVIDED “AS IS” AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE HELMA PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Helma includes third party software released under different specific license terms. See the licenses directory in the Helma distribution for a list of these license.

Sarissa

Copyright © 2004 Manos Batsis.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

ICU

ICU License - ICU 1.8.1 and later COPYRIGHT AND PERMISSION NOTICE Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

Sun's JAXB Implementation – JSDK 1.5 relaxngDatatype.jar 1.0 License

Copyright © 2001, Thai Open Source Software Center Ltd, Sun Microsystems. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

W3C IPR SOFTWARE NOTICE

Copyright © 2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

Note: The original version of the W3C Software Copyright Notice and License could be found at <http://www.w3.org/Consortium/Legal/copyright-software-19980720>.

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

Contents

General Preface

- About This PeopleBookxxv**
- PeopleSoft Enterprise Application Prerequisites.....xxv
- Application Fundamentals.....xxv
- Documentation Updates and Printed Documentation.....xxvi
 - Obtaining Documentation Updates.....xxvi
 - Downloading and Ordering Printed Documentation.....xxvi
- Additional Resources.....xxvii
- Typographical Conventions and Visual Cues.....xxviii
 - Typographical Conventions.....xxviii
 - Visual Cues.....xxix
 - Country, Region, and Industry Identifiers.....xxix
 - Currency Codes.....xxx
- Comments and Suggestions.....xxx
- Common Elements Used in PeopleBooks.....xxx

Preface

- PeopleSoft Enterprise Advanced Configurator for CRM Preface.....xxxiii**
- PeopleSoft Enterprise Advanced Configurator 9 PeopleBook.....xxxiii
- PeopleSoft Enterprise Order Capture 9 PeopleBook.....xxxiv

Part 1 Getting Started

Chapter 1

- Getting Started with PeopleSoft Enterprise Advanced Configurator.....3**
- Additional Documentation for Advanced Configurator.....3
- Testing and Administration Tools.....3
- Advanced Configurator Architecture.....4
- Configurator Interfaces.....4
- Advanced Configurator Implementation.....5
 - Implementing an Integrated Configurator Solution.....6
 - Implementing a Standalone Configurator Solution.....6

Chapter 2
Other Sources of Information.....7

Part 2
Product Modeling with a Component Model

Chapter 3
Understanding Modeling..... 11

Basic Model Concepts..... 11

 Visual Modeler..... 11

 Visual Modeler Objects..... 12

 Domain Members..... 14

 Selection Points..... 14

Relationships Between Objects..... 14

 Requirement Constraint..... 18

 Dynamic Default..... 19

 Resource Constraint..... 20

 Summation..... 20

 Elimination..... 21

 Comparison..... 21

 Effectivity Dates..... 24

Expressions in Relationships..... 25

Relationship Explanations..... 27

Relationship Properties..... 28

Default Values Within Expressions..... 31

Quantities in Modeling..... 32

 Static Default Quantities..... 34

 Dynamic Default Quantities..... 34

 Multiple Selections on a Single Domain Member..... 34

 Understanding Minimum and Maximum Selections and Limits..... 40

 Domain Member Min/Max..... 41

 Interaction between Default Quantities and Min/Max Settings at Run Time..... 42

 Minimum Violation Explanation and Incomplete Configuration Explanation..... 43

Creating Parameterized Explanations..... 44

Optimizing Performance and Minimizing Model Maintenance..... 45

Using the Sample Models..... 46

The Modeling Process..... 46

Model Tester..... 47

Interfacing with Third-Party Tools.....47

 Microsoft SQL Server, Oracle Databases, and IBM DB2.....47

 MacroMedia DreamWeaver.....47

 Source Control Interfaces.....47

Chapter 4

Setting Up the Modeling Environment.....49

Common Elements in this Chapter.....49

Connecting to Third-Party Software.....50

 Source Control Software.....50

 Database Interface Configuration.....51

 Configuring JNDIDBName.properties.....53

Connecting to a Database from Visual Modeler.....54

 Specify a Database Connection.....54

 Specify a Default Database.....54

Getting Started with Visual Modeler.....55

 Model Structure View.....55

 Components and Files View.....56

 Properties Editor.....56

 Overview Window.....57

 Find Window.....57

Understanding Project Files.....57

 Creating a New Project or Workspace.....59

 Specifying Model Project Settings.....60

 Adding a Project to Source Control.....62

Importing and Exporting Models.....63

 Exporting a Model.....63

 Importing a Model.....64

Compiling a Model.....65

Using the Model Tester.....66

Internalizing Model Data.....67

Chapter 5

Creating Objects for the Model.....69

Creating a Class.....69

Deleting a Class.....70

Changing Class Structure.....71

Adding Class Attributes.....72

Creating Internal Domain Members.....	73
Creating a “None” Domain Member.....	73
Assigning Values to Attributes.....	74
Inputting Date-Type Attributes Manually.....	74
Setting Up Binding for External Domain Members.....	75
Selecting a Primary Table.....	75
Filtering and Manipulating Table Data.....	80
Storing a Dynamic Default Quantity in a Database.....	80
Retrieving Expression Values and Externs from a Database.....	81
Working with Selection Points.....	81
Internalizing Data.....	85

Chapter 6

Creating Relationships Between Model Objects.....	87
Prerequisites.....	87
Common Elements Used in this Chapter.....	87
Creating and Editing Expressions.....	89
Creating Externs.....	91
Creating a Relationship.....	93
Working with Relationships.....	96
Editing Compatibility Constraints.....	98
Editing Requirement Constraints.....	99
Editing Dynamic Defaults.....	100
Editing Resource Constraints.....	101
Editing Summation Relationships.....	102
Editing Elimination Constraints.....	103
Editing Comparison Constraints.....	104
Creating Relationships Outside the Model with SQL Queries.....	105

Chapter 7

Specifying Quantities on Selection Points.....	109
Understanding Quantity Setup.....	109
Specifying the Number of Allowed Selections and Optional or Required Status.....	110
Specifying Single- or Multi-Select Controls.....	111
Setting Quantity Limits on Domain Members.....	112
Setting Default Selections and Quantities.....	114
Setting Explicit Default Choices and Quantities.....	115
Getting Default Selections and Quantities at Runtime Through Attributes.....	116

Defining a Dynamic Default Quantity for a Selection.....	117
Attaching Metadata to Selection Points.....	118

Part 3

Product Modeling with Compound Models

Chapter 8

Understanding Compound Modeling.....	123
Applications for Compound Models.....	123
Compound Model Structure Types.....	123
Architecture.....	126
Relationships in a Compound Model.....	127
Modeling Strategy.....	128

Chapter 9

Working with Compound Models.....	129
Getting Started with Compound Models.....	129
Creating a Compound Modeling Project.....	130
Editing Project Settings.....	131
Creating a Configurable Component.....	132
Deleting a Configurable Component.....	134
Rearranging Component Models in the Compound Model.....	135
Adding and Removing a Component Model from the Project.....	135
Editing Default Values.....	137
Creating and Deleting Relationships Between Configurable Components.....	138
Displaying a Compound Model Relationship.....	141
Specifying Required Relationships.....	142
Editing Component Model Versions.....	144
Compiling, Running, and Testing a Compound Model.....	146
Managing Simultaneous Model Development Among Team Members.....	146

Chapter 10

Standardizing Compound ModelBuilding.....	147
Creating and Editing Configurable Component Types.....	147
Creating and Editing Connection Point Types.....	150

Part 4 Application Extensions

Chapter 11

Client Operations Processor API.....	155
Understanding the COP Java API.....	155
Choices.....	155
Decision Points and Domain Members.....	156
Application Classes.....	157
ClientOperations.....	157
Configuration.....	158
ControlData.....	158
ControllItem.....	158
Choice.....	158
DMChoice.....	159
EVChoice.....	159
ItemFilter.....	159
ItemIterator.....	159
ExternVar.....	159
NumericData.....	160
Violation.....	160

Chapter 12

Using the COP Java API.....	161
ClientOperations.....	161
Methods.....	161
Initializing the COP.....	162
Releasing the COP.....	164
Processing and Displaying a Page.....	164
Getting a ControlData Object.....	165
Specifying Delta-Pricing and Total-Pricing Requirements.....	168
Getting Other Display Information.....	169
Verifying a Configuration.....	169
Configuration.....	169
Methods.....	170
Saving and Restoring a Configuration.....	170
ControlData.....	171
Methods.....	172

Getting Display Information for a Decision Point and Its Domain Members.....172

Getting the State of a Decision Point.....173

Sorting and Filtering.....174

Handling Deleted Domain Members.....176

ControllItem.....176

 Methods.....176

 Getting Display Information for a Domain Member.....176

 Getting the State of a Domain Member.....177

Choice.....180

DMChoice.....180

 Methods.....181

 Examining a DMChoice.....181

EVChoice.....181

 Methods.....181

 Examining an EVChoice.....182

ItemFilter.....182

 Methods.....182

 Filtering Out Domain Members.....182

ItemIterator.....183

ExternVar.....183

NumericData.....183

Violation.....184

Chapter 13

Understanding the Configurator XML Interface.....185

Request-Response.....185

Elements and Attributes.....185

Retrieving Model Information.....186

Updating a Configuration Interactively.....186

Retrieving Configuration Information.....187

Copying a Configuration.....187

Using Batch Configuration Mode.....188

Changing the Order Status of a Configuration.....188

COP.dtd.....188

Element-Attribute Trees.....188

Chapter 14

Retrieving Model Information.....189

Elements and Attributes.....	189
Version and Compile Version.....	189
Latest Version and Compile Version.....	189
Latest Compile Version.....	190
Error Messages.....	191
Decision Points.....	192
All Decision Points.....	192
Public Decision Points.....	193

Chapter 15

Updating a Configuration.....	195
Updating a Configuration.....	195
Elements and Attributes.....	196
Choices.....	196
Choices and Response.....	197

Chapter 16

Retrieving Configuration Information.....	201
Understanding Configuration Information.....	201
Elements and Attributes.....	202
Total Price.....	203
Choices.....	203
Domain Member Data.....	204
Every Decision Point.....	204
Selected Decision Points.....	206
Sorting Domain Members.....	207
Filtering Domain Members.....	209
Explanations.....	209
Attributes.....	212
Delta Price.....	217
Class.....	218
State and Quantity.....	219
Multi-Select Decision Points.....	220
Global Explanations.....	221
Global Only.....	221
Global and Decision Point.....	222
Numeric Values.....	223
All Values.....	223

Selected Values.....223
 Value (VL).....224

Chapter 17

Retrieving Saved Configuration Information.....229
 Understanding Saved Configuration Information.....229
 Elements and Attributes.....230
 The CONFIGURATION Element.....231
 The CONFIG_DETAILS Element.....232
 The DELTA_INFO Element.....233
 Components.....233
 Compounds.....234
 The SECTION Element.....235
 Total Price.....236
 Compound Violations.....236
 Components.....237
 Choices.....242
 Choice Violations.....249
 Component Violations.....250
 Externs.....250
 Numeric Values.....252
 External Variables.....253
 All Values.....254
 Selected Values.....254
 Configuration Attributes.....255
 Hierarchical Component Structure.....256
 Connections.....258
 Completeness Information.....259
 Summary of Configuration Information Elements and Attributes.....260

Chapter 18

Copying a Configuration.....263
 Elements and Attributes.....263
 Copy and Response.....263

Chapter 19

Using Batch Configuration Mode.....265

Elements and Attributes.....	265
Configuring a Component.....	265
Configuring a Compound Configuration.....	266
Saving a Configuration.....	267
Retrieving a Configuration.....	267

Chapter 20

Changing the Order Status of a Configuration.....	269
Elements and Attributes.....	269
Order Change and Response.....	269

**Part 5
PeopleSoft CRM Order Capture Integration**

Chapter 21

Understanding Integration with PeopleSoft CRM Order Capture.....	273
Integration with PeopleSoft Enterprise Order Capture Applications.....	273
Insurance and Financial Products.....	273
Service Products.....	274
Security.....	274

Chapter 22

Setting Up Integration.....	275
Setting Up PeopleSoft Advanced Configurator for Integration.....	275
Setting Up PeopleSoft CRM to Integrate with Advanced Configurator.....	276
Page Used to Set Up Configurator Integration with PeopleSoft CRM.....	277
Associating Advanced Configurator Messaging Node and Enabling Debugging.....	277
Creating Advanced Configurator Schemas.....	278
Pages Used to Create Advanced Configurator Schemas.....	279
Understanding Configurator Schemas.....	279
Creating Schemas for External Solutions.....	280
Creating Schemas for Internal Solutions.....	281
Establishing Configuration Display and Pricing Options.....	284
Specifying Request Details.....	287
Accessing the Advanced Configurator Solution from Within PeopleSoft CRM.....	290
Understanding How to Access Advanced Configurator.....	290

Page Used to Access the Advanced Configurator Solution from Within PeopleSoft CRM.....	291
Sample Product Configuration.....	291
Viewing Configuration Details.....	292

Part 6

Building a Custom User Interface

Chapter 23

Understanding the Runtime System.....	297
Deployment Framework.....	297
Advanced Configurator Web Components.....	300
Sequential Application JSP Pages.....	301
Deployment for a Web Application Based on a Single Component Model.....	302
Optimizing Performance.....	302
Restore Policy.....	303
Deployment for a Solution in a Mobile Environment.....	303
Deployment for a Web Application Based on a Compound Model.....	304

Chapter 24

JSP and Page Templates.....	307
The Midtier Framework.....	307
Scope of the Servlet.....	309
Using JSP Processing.....	309
Writing JSP.....	310
Using Generated Java and Class Files.....	311

Chapter 25

Processing User Picks and Entries.....	313
Understanding Runtime Processing.....	313
Initializing the WCP.....	314
Processing User Picks and Entries.....	315
Configuration Records.....	315
Attribute Records.....	316
Making COP Calls.....	316
Using WCP Methods.....	317
Getting Decision or Selection Points.....	317

Getting and Processing Stored Configuration Records.....	317
Getting Model Name, Version, and Compile Version.....	317
Clearing Model State.....	318
Releasing the WCP.....	318

Chapter 26

Processing Configurator Form Controls in JSP Pages.....	319
Understanding Configurator Form Control Processing.....	319
Configurator JSP Page Flow.....	319
Processing Configurator Form Controls.....	320
Pre-Process Form Page.....	320
Process Form Page.....	321
Constants Page.....	321
Start Form Page.....	321
Control Page.....	322
End Form Page.....	322
Using Configurator JSP Pages in a Solution.....	322

Chapter 27

Using JSP Form Control Templates.....	325
Understanding Form Control Templates.....	325
Understanding Properties, Parameters, and Attributes.....	326
Properties.....	326
Parameters.....	326
Attributes.....	326
Understanding JSP Code Templates.....	326
Using Configuration Form Control Templates.....	341
Plugging Form Controls into the Application Pages.....	342
Parameters in the Inclusion Set.....	344
Specifying the Model and Locale Properties for the Solution.....	347
Specifying Solution Information Properties.....	349
Specifying Display Properties.....	350
Displaying Delta Information.....	350
Displaying Delta Pricing.....	351
Application Page Example.....	352
Configuring a Form Control Template.....	353
Registering Custom Form Control Templates.....	354
Custom Form Control Template Example.....	355

Common Errors.....	356
--------------------	-----

Chapter 28

Using the Page Editor Extensions for Dreamweaver.....	359
Understanding Dreamweaver Extensions.....	359
Advanced Configurator Runtime Objects.....	359
Creating a Solution.....	361
Editing CalicoUI.properties.....	361
Inserting a Configurator Runtime Object.....	362
Inserting a Form.....	362
Inserting a Button.....	363
Inserting a List.....	363
Inserting a Group.....	365
Inserting a Table.....	367
Inserting an Image.....	368
Inserting Why Help.....	371
Inserting a Numeric Data Object.....	372
Editing Properties of Advanced Configurator Objects.....	374
Editing Forms and Buttons.....	374
Editing Lists, Groups, and Tables.....	374

Chapter 29

Compound Modeling.....	375
Understanding the Compound Model at Run Time.....	375
Runtime Capabilities.....	375
Architecture.....	376
Using Compound Model JSP Pages.....	377
Calling the Compound Model API.....	379
Creating an Application from the Sample.....	380
Viewing the Sample Application.....	380
Node-Hub-Circuit Services.....	381
Configurable Components.....	381
Relationships.....	382
Modeling Node-Hub-Circuit Services.....	383
Configuring Node-Hub-Circuit Services.....	383
Creating a Compound Configuration.....	383
Reconfiguring a Compound Configuration.....	383
Obtaining the Configuration Delta.....	383

Part 7 Mobile Product Configuration

Chapter 30

Understanding Mobile Product Configuration.....	387
Mobile Configuration.....	387
Solution Package Contents.....	388
Functional Components.....	388
Data.....	388
Model Data.....	389
Site Support Data.....	389
Configuration Data.....	389
Solution Distribution and Installation.....	389
Types of Models.....	390
Mobile Configurator Packaging Tool.....	390
Mobile Machine Requirements.....	393
Package Maintenance.....	393
Controlling Field Version Use.....	393
Guidelines for Maintenance.....	393

Chapter 31

Mobile Solution Administration.....	395
Preparing a Solution for Packaging.....	395
Internalizing Model Data.....	396
Setting Configuration Restore Policy.....	397
Localizing Solution UI Files.....	398
Customizing and Localizing the Mobile Configurator Client.....	398
Customizing the Mobile Client.....	398
Localizing the Mobile Client.....	399
Integrating Mobile Configurator with ERP Systems.....	400
MTS Interfaces.....	400
Sample Implementations.....	401
Using the Transfer Service.....	401
Using Source Control.....	402
Integrating Mobile Configurator with Microsoft COM-Compatible Applications.....	402

Chapter 32

Creating and Maintaining Mobile Solutions.....	403
Understanding the Packaging Tool.....	403
Setting the Application Root.....	404
Creating a New Solution Package.....	405
Adding a Model to the Solution Package.....	407
Setting End-User Installation Options.....	407
Modifying a Solution Package.....	408
Adding Non-Solution Files to the Solution Package.....	408
Accessing Package Properties.....	408
Adding Annotation to the Package.....	408
Generating the Solution Package Installer.....	408
Updating a Solution Package Installer.....	409
Updating Only the Model(s) of a Package.....	409
Printing Solution Package Content.....	410

Part 8**Advanced Configurator System Administration****Chapter 33**

Understanding Advanced Configurator Administration.....	413
--	------------

Chapter 34

Using Administration Tools.....	415
Administration Console.....	415
Testing Solutions.....	418
Page Used to Test Solutions.....	418
Accessing the Solution Tester.....	421
Understanding the Output and Solution User Interface.....	422
Setting Configuration Solution Parameters.....	422
Model Tester.....	424

Chapter 35

Maintaining the Advanced Configurator System.....	425
Managing Model Versioning.....	425
Loading Models.....	426

Managing the Memory Usage of the Configurator Server.....	427
Compressing Configuration Data.....	428
Using the Explanations.properties File.....	429
Copying the Explanations.properties File.....	429
Searching for the Explanations.properties File.....	429
Compiling Models from the Command Line.....	430
Accessing and Using COPXML Servlet Statistics.....	430

Appendix A

Visual Modeler Expression Editor Functions.....	433
Numeric Operators and Functions.....	433
Boolean Functions.....	436
Date Functions.....	438
String Functions.....	441

Appendix B

Creating and Adding User-Defined Functions.....	443
Adding a User-Defined Function.....	443
Implementing the UserFunction Interface.....	444
Methods.....	444
Exceptions.....	445
Editing UserFunctions.xml.....	446
Using the Sample User-Defined Function getQuantity().....	448
Understanding the getQuantity() Sample Function.....	448
Setting Up getQuantity().....	449
Viewing getQuantity() Behavior.....	450

Appendix C

Advanced Configurator Form Controls.....	453
Single-Select Group Form Control.....	453
Multi-Select Group Form Control.....	454
Single-Select Table Form Control.....	455
Multi-Select List Form Control.....	456
Single-Select List Form Control.....	456
Multi-Select Table Form Control.....	457
Single-Select Image.....	458
Single-Select Image Table.....	458

Application Why Help.....	460
Form Control Why Help.....	460
Text Input Form Control.....	461
Numeric Data Form Control.....	461
Extern Entry.....	462

Appendix D

Compound Model Properties File.....	463
Properties Description.....	463
File Text.....	464

Appendix E

Node-Circuit-Hub Service.....	467
Description of Services.....	467
XML Representation of Compound Structure Definition.....	468

Appendix F

Mobile Configurator COM API.....	471
PeopleSoft.MCfmgProxy Methods.....	471
Start.....	471
Stop.....	471
Show.....	471
Hide.....	472
NewConfiguration(String sSolutionID).....	472
ReConfigure(String sXML).....	472
WaitForConfiguration(Integer nTimeout).....	473
IsConfigurationReady.....	473
GetConfiguration.....	473
SetLabelToString(Integer nLabelID, String NewLabel).....	474
StartCountdown(Integer nCountdownTimeInMins, Integer nWarnLeftTimeInMins, String strWarnMessage, String strContinueLabel, String strStopLabel).....	474
StopCountdown.....	475
Sample Application Client Code.....	475

Appendix G

PCIF.....	477
MODEL Element.....	477

DATABASE_REFERENCE Element.....	479
CLASS Element.....	480
CLASS_ATTRIBUTE Element.....	481
DEFAULT_VALUE Element.....	482
DOMAIN_MEMBER Element.....	482
DM_ATTRIBUTE Element.....	483
STANDARD_QUERY Element.....	483
PRIMARY_TABLE Element.....	484
COLUMN Element.....	484
WHERE Element.....	485
SECONDARY_TABLE Element.....	485
JOIN Element.....	485
ADVANCED_QUERY Element.....	485
QUERY_TEXT Element.....	486
SELECTION_POINT Element.....	486
STATIC_DEFAULTS Element.....	488
STATIC_DEFAULT Element.....	489
SELECTION_POINT_MIN_QTY_SETTINGS Element.....	489
DOMAIN_MEMBER_MIN_QTY_SETTINGS Element.....	490
DOMAIN_MEMBER_MAX_QTY_SETTINGS Element.....	491
DYNAMIC_DEFAULT Element.....	491
EXPLANATION Element.....	493
EXPRESSION Element.....	493
NOT_COMPATIBLE Element.....	493
EXTERN Element.....	494
EFFECTIVITY Element.....	495
EFFECTIVEDATE Element.....	495
COMMENT Element.....	496
ARGUMENT Element.....	496
RHS_ARGUMENT Element.....	496
ROW Element.....	497
SET Element.....	497
VALUE Element.....	498
The CONSTRAINT_QUERY Element.....	498
SQL_CLAUSE Element.....	498
COMPATIBLE Element.....	498
REQUIRED Element.....	500
ELIMINATION Element.....	501
COMPARISON Element.....	502
RESOURCE_CONSTRAINT Element.....	503

RESOURCE_PROVIDERS Element.....504
 ATTRIBUTE Element.....504
 RESOURCE_CONSUMERS Element.....505
 SELECTION_POINT_ATTRIBUTE Element.....505
 SUMMATION Element.....506
 SUMMANDS Element.....506
 TOTAL_ATTRIBUTE Element.....507

Appendix H

Element-Attribute Trees.....509
 Complete COP XML.....509
 Without Attributes.....509
 With Attributes.....510
 Configurator XML Interface.....511
 Request.....511
 Response.....511

Glossary of PeopleSoft Enterprise Terms.....513

Index535

About This PeopleBook

PeopleSoft Enterprise PeopleBooks provide you with the information that you need to implement and use PeopleSoft Enterprise applications from Oracle.

This preface discusses:

- PeopleSoft Enterprise application prerequisites.
- Application fundamentals.
- Documentation updates and printed documentation.
- Additional resources.
- Typographical conventions and visual cues.
- Comments and suggestions.
- Common elements in PeopleBooks.

Note. PeopleBooks document only elements, such as fields and check boxes, that require additional explanation. If an element is not documented with the process or task in which it is used, then either it requires no additional explanation or it is documented with common elements for the section, chapter, PeopleBook, or product line. Elements that are common to all PeopleSoft Enterprise applications are defined in this preface.

PeopleSoft Enterprise Application Prerequisites

To benefit fully from the information that is covered in these books, you should have a basic understanding of how to use PeopleSoft Enterprise applications.

You might also want to complete at least one introductory training course, if applicable.

You should be familiar with navigating the system and adding, updating, and deleting information by using PeopleSoft Enterprise menus, pages, or windows. You should also be comfortable using the World Wide Web and the Microsoft Windows or Windows NT graphical user interface.

These books do not review navigation and other basics. They present the information that you need to use the system and implement your PeopleSoft Enterprise applications most effectively.

Application Fundamentals

Each application PeopleBook provides implementation and processing information for your PeopleSoft Enterprise applications.

For some applications, additional, essential information describing the setup and design of your system appears in a companion volume of documentation called the application fundamentals PeopleBook. Most product lines have a version of the application fundamentals PeopleBook. The preface of each PeopleBook identifies the application fundamentals PeopleBooks that are associated with that PeopleBook.

The application fundamentals PeopleBook consists of important topics that apply to many or all PeopleSoft Enterprise applications. Whether you are implementing a single application, some combination of applications within the product line, or the entire product line, you should be familiar with the contents of the appropriate application fundamentals PeopleBooks. They provide the starting points for fundamental implementation tasks.

Documentation Updates and Printed Documentation

This section discusses how to:

- Obtain documentation updates.
- Download and order printed documentation.

Obtaining Documentation Updates

You can find updates and additional documentation for this release, as well as previous releases, on Oracle's PeopleSoft Customer Connection website. Through the Documentation section of Oracle's PeopleSoft Customer Connection, you can download files to add to your PeopleBooks Library. You'll find a variety of useful and timely materials, including updates to the full line of PeopleSoft Enterprise documentation that is delivered on your PeopleBooks CD-ROM.

Important! Before you upgrade, you must check Oracle's PeopleSoft Customer Connection for updates to the upgrade instructions. Oracle continually posts updates as the upgrade process is refined.

See Also

Oracle's PeopleSoft Customer Connection, http://www.oracle.com/support/support_peoplesoft.html

Downloading and Ordering Printed Documentation

In addition to the complete line of documentation that is delivered on your PeopleBook CD-ROM, Oracle makes PeopleSoft Enterprise documentation available to you via Oracle's website. You can:

- Download PDF files.
- Order printed, bound volumes.

Downloading PDF Files

You can download PDF versions of PeopleSoft Enterprise documentation online via the Oracle Technology Network. Oracle makes these PDF files available online for each major release shortly after the software is shipped.

See Oracle Technology Network, <http://www.oracle.com/technology/documentation/psftent.html>.

Ordering Printed, Bound Volumes

You can order printed, bound volumes of selected documentation via the Oracle Store.

See Oracle Store, http://oraclestore.oracle.com/OA_HTML/ibeCCtpSctDspRte.jsp?section=14021

Additional Resources

The following resources are located on Oracle's PeopleSoft Customer Connection website:

Resource	Navigation
Application maintenance information	Updates + Fixes
Business process diagrams	Support, Documentation, Business Process Maps
Interactive Services Repository	Support, Documentation, Interactive Services Repository
Hardware and software requirements	Implement, Optimize + Upgrade; Implementation Guide; Implementation Documentation and Software; Hardware and Software Requirements
Installation guides	Implement, Optimize + Upgrade; Implementation Guide; Implementation Documentation and Software; Installation Guides and Notes
Integration information	Implement, Optimize + Upgrade; Implementation Guide; Implementation Documentation and Software; Pre-Built Integrations for PeopleSoft Enterprise and JD Edwards EnterpriseOne Applications
Minimum technical requirements (MTRs)	Implement, Optimize + Upgrade; Implementation Guide; Supported Platforms
Documentation updates	Support, Documentation, Documentation Updates
PeopleBooks support policy	Support, Support Policy
Prerelease notes	Support, Documentation, Documentation Updates, Category, Release Notes
Product release roadmap	Support, Roadmaps + Schedules
Release notes	Support, Documentation, Documentation Updates, Category, Release Notes
Release value proposition	Support, Documentation, Documentation Updates, Category, Release Value Proposition
Statement of direction	Support, Documentation, Documentation Updates, Category, Statement of Direction
Troubleshooting information	Support, Troubleshooting
Upgrade documentation	Support, Documentation, Upgrade Documentation and Scripts

Typographical Conventions and Visual Cues

This section discusses:

- Typographical conventions.
- Visual cues.
- Country, region, and industry identifiers.
- Currency codes.

Typographical Conventions

This table contains the typographical conventions that are used in PeopleBooks:

Typographical Convention or Visual Cue	Description
Bold	Indicates PeopleCode function names, business function names, event names, system function names, method names, language constructs, and PeopleCode reserved words that must be included literally in the function call.
<i>Italics</i>	Indicates field values, emphasis, and PeopleSoft Enterprise or other book-length publication titles. In PeopleCode syntax, italic items are placeholders for arguments that your program must supply. We also use italics when we refer to words as words or letters as letters, as in the following: Enter the letter <i>O</i> .
KEY+KEY	Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For ALT+W, hold down the ALT key while you press the W key.
Monospace font	Indicates a PeopleCode program or other code example.
“ ” (quotation marks)	Indicate chapter titles in cross-references and words that are used differently from their intended meanings.
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe ().

Typographical Convention or Visual Cue	Description
[] (square brackets)	Indicate optional items in PeopleCode syntax.
& (ampersand)	<p>When placed before a parameter in PeopleCode syntax, an ampersand indicates that the parameter is an already instantiated object.</p> <p>Ampersands also precede all PeopleCode variables.</p>

Visual Cues

PeopleBooks contain the following visual cues.

Notes

Notes indicate information that you should pay particular attention to as you work with the PeopleSoft Enterprise system.

Note. Example of a note.

If the note is preceded by *Important!*, the note is crucial and includes information that concerns what you must do for the system to function properly.

Important! Example of an important note.

Warnings

Warnings indicate crucial configuration considerations. Pay close attention to warning messages.

Warning! Example of a warning.

Cross-References

PeopleBooks provide cross-references either under the heading “See Also” or on a separate line preceded by the word *See*. Cross-references lead to other documentation that is pertinent to the immediately preceding documentation.

Country, Region, and Industry Identifiers

Information that applies only to a specific country, region, or industry is preceded by a standard identifier in parentheses. This identifier typically appears at the beginning of a section heading, but it may also appear at the beginning of a note or other text.

Example of a country-specific heading: “(FRA) Hiring an Employee”

Example of a region-specific heading: “(Latin America) Setting Up Depreciation”

Country Identifiers

Countries are identified with the International Organization for Standardization (ISO) country code.

Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in PeopleBooks:

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

Industries are identified by the industry name or by an abbreviation for that industry. The following industry identifiers may appear in PeopleBooks:

- USF (U.S. Federal)
- E&G (Education and Government)

Currency Codes

Monetary amounts are identified by the ISO currency code.

Comments and Suggestions

Your comments are important to us. We encourage you to tell us what you like, or what you would like to see changed about PeopleBooks and other Oracle reference and training materials. Please send your suggestions to your product line documentation manager at Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, U.S.A. Or email us at appsdoc@us.oracle.com.

While we cannot guarantee to answer every email message, we will pay careful attention to your comments and suggestions.

Common Elements Used in PeopleBooks

As of Date	The last date for which a report or process includes data.
Business Unit	An ID that represents a high-level organization of business information. You can use a business unit to define regional or departmental units within a larger organization.
Description	Enter up to 30 characters of text.
Effective Date	The date on which a table row becomes effective; the date that an action begins. For example, to close out a ledger on June 30, the effective date for the ledger closing would be July 1. This date also determines when you can view and change the information. Pages or panels and batch processes that use the information use the current row.

Once, Always, and Don't Run	<p>Select Once to run the request the next time the batch process runs. After the batch process runs, the process frequency is automatically set to Don't Run.</p> <p>Select Always to run the request every time the batch process runs.</p> <p>Select Don't Run to ignore the request when the batch process runs.</p>
Process Monitor	<p>Click to access the Process List page, where you can view the status of submitted process requests.</p>
Report Manager	<p>Click to access the Report List page, where you can view report content, check the status of a report, and see content detail messages (which show you a description of the report and the distribution list).</p>
Request ID	<p>An ID that represents a set of selection criteria for a report or process.</p>
Run	<p>Click to access the Process Scheduler request page, where you can specify the location where a process or job runs and the process output format.</p>
SetID	<p>An ID that represents a set of control table information, or TableSets. TableSets enable you to share control table information and processing options among business units. The goal is to minimize redundant data and system maintenance tasks. When you assign a setID to a record group in a business unit, you indicate that all of the tables in the record group are shared between that business unit and any other business unit that also assigns that setID to that record group. For example, you can define a group of common job codes that are shared between several business units. Each business unit that shares the job codes is assigned the same setID for that record group.</p>
Short Description	<p>Enter up to 15 characters of text.</p>
User ID	<p>An ID that represents the person who generates a transaction.</p>

PeopleSoft Enterprise Advanced Configurator for CRM Preface

This preface discusses:

- *PeopleSoft Enterprise Advanced Configurator 9 PeopleBook*.
- *PeopleSoft Enterprise Order Capture 9 PeopleBook*.

PeopleSoft Enterprise Advanced Configurator 9 PeopleBook

The *PeopleSoft Enterprise Advanced Configurator 9 PeopleBook* provides implementation and processing information for the Advanced Configurator application. Information is organized into these topics:

- *Getting Started* provides an overview of the application's basic concepts, capabilities, implementation phases, and tools.
- *Product Modeling with a Component Model* explains how to build the central component of a configuration implementation, the model, to represent a product or service, using PeopleSoft Visual Modeler.
- *Product Modeling with a Compound Model* explains how to represent a product or service with parts that are themselves configurable.
- *Application Extensions* describes how to build specific functionality by calling or extending Advanced Configurator Java APIs.
- *PeopleSoft Enterprise Order Capture Integration* describes how to set up Advanced Configurator for configuration sessions from within the PeopleSoft Enterprise Order Capture application.
- *Building a Custom User Interface* details the front-end components that allow you to build a completely custom configuration interface for hosting on an independent production environment.
- *Mobile Product Configuration* explains the capabilities of the mobile configuration components and their setup and maintenance for a disconnected field user base.
- *Advanced Configurator System Administration* describes how to use the deployment and testing tools to validate and maintain configuration implementations.

In addition to the core reference sections of this PeopleBook, there are six appendixes:

- *Visual Modeler Expression Editor Functions* lists the many operators and functions available for writing expressions to describe model behavior.
- *Creating and Adding User-Defined Functions* is a tutorial for writing your own functions for use in creating expressions for a model.
- *Advanced Configurator Form Controls* provides a reference for the HTML and JSP code that renders each of the form controls available for a custom configuration interface.
- *Compound Model Properties File* provides a hard-copy reference for the compound model properties file in two versions: annotated and non-annotated.
- *Node-Circuit-Hub Service Sample Model* describes a sample complex product (communications services) offering that includes the XML representation of its compound structure definition

- *Mobile Configurator COM API* is a developer's reference for the classes and methods that allow a Mobile Order Capture user to initiate a configuration session with Mobile Configurator.

There is a glossary of terms at the end of the book.

PeopleSoft Enterprise Order Capture 9 PeopleBook

Additional essential information describing the setup and design of your system appears in a companion volume of documentation called *PeopleSoft Enterprise Order Capture 9 PeopleBook*.

The *PeopleSoft Enterprise Order Capture Applications 9 PeopleBook* consists of topics that apply if you are integrating PeopleSoft Enterprise Order Capture with Advanced Configurator.

PART 1

Getting Started

Chapter 1

Getting Started with PeopleSoft Enterprise Advanced Configurator

Chapter 2

Other Sources of Information

CHAPTER 1

Getting Started with PeopleSoft Enterprise Advanced Configurator

This chapter provides an overview of Advanced Configurator architecture and discusses:

- Additional documentation for Advanced Configurator.
- Testing and administration tools.
- Implementing a configuration solution.
- Other sources of information.

Additional Documentation for Advanced Configurator

This section lists the documentation for PeopleSoft Advanced Configurator that is available in the Configurator installation and on PeopleSoft Customer Connection.

See Also

PeopleSoft Enterprise CRM 9 Installation Guide

PeopleSoft Advanced Configurator API Reference Guide

Testing and Administration Tools

This section discusses the tools available for testing and administration.

Configurator includes two tools to help you validate your model and its associated solution. The *Solution Tester* and the *Model Tester* provide valuable troubleshooting information. Solutions require updating to reflect changes in the product or service.

Note. The *Administration Tool* can help you track and manage versions as well as service, compile, and run models remotely.

See Also

Part 8, “Advanced Configurator System Administration,” page 411

Advanced Configurator Architecture

This section describes the basic elements of Advanced Configurator architecture:

- Application server
- Web server
- Relational databases

Application Server

The application server, such as BEA WebLogic , has three "tiers" within it: dynamic presentation logic, business logic, and database abstraction.

- Dynamic presentation

This tier is used to process any presentation content that is determined at run time. It is conditional based upon user actions and selections, and can be personalized.

- Business logic

This tier houses the Configurator models and other product components.

- Database abstraction

This tier contains calls to the database so that designers do not need to know which specific database is being used. It also maintains connections to any databases that might be utilized.

After the dynamic content is processed by these "tiers," it is then passed to the web server for display in the browser.

Web Server

Web servers are used as the static presentation tier. A web server displays static HTML pages and images. It also displays the HTML results of the processed Java Server Pages (JSPs) after the application and database servers have compiled the dynamic information.

Relational Databases

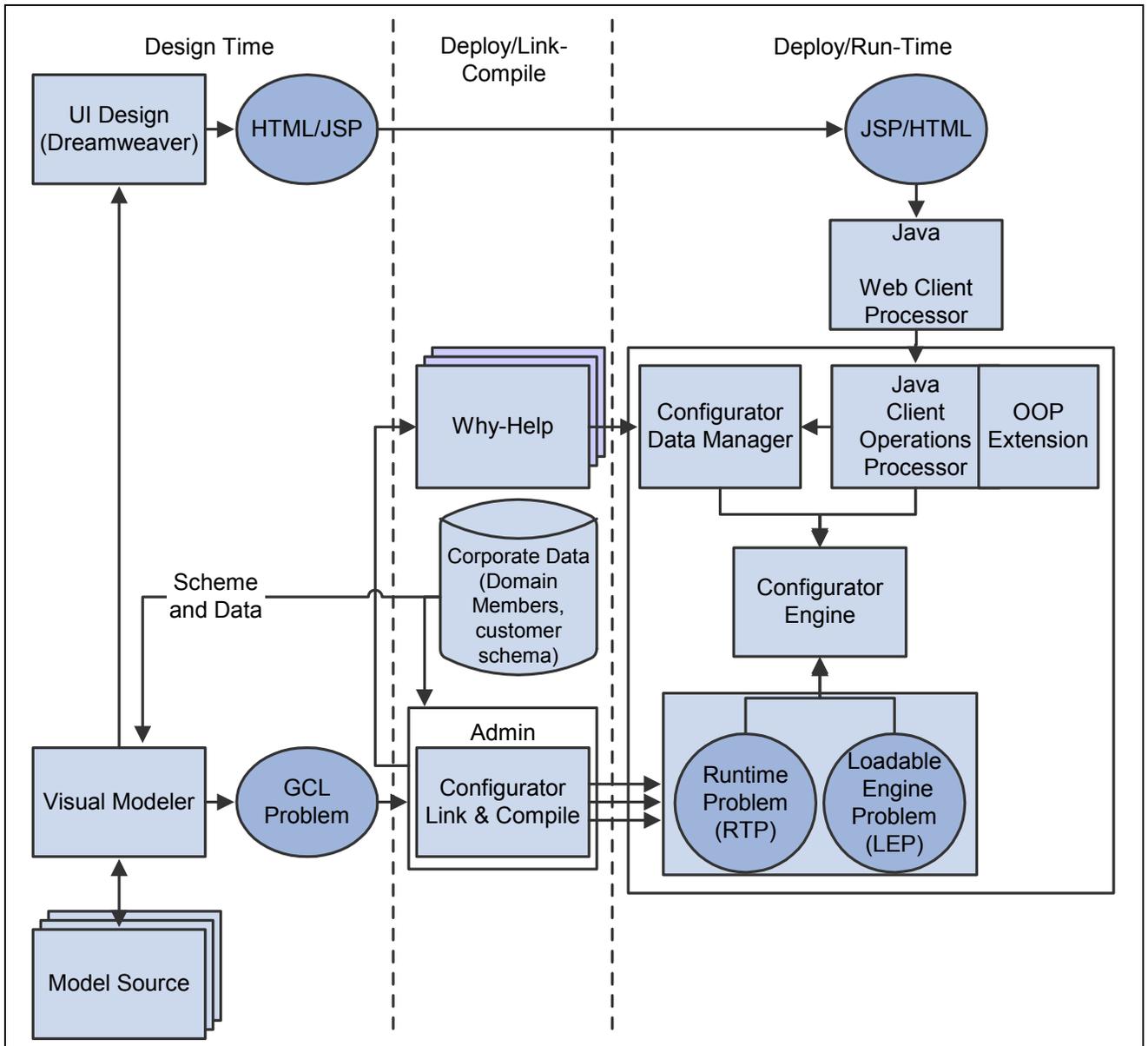
The relational databases contain any external data that you use in product models. Examples of these might be third-party product tables, catalogs, or marketing information. This data can be refreshed without requiring business logic to change. This allows for a flexible and maintainable system.

Configurator Interfaces

The PeopleSoft Advanced Configurator interacts with industry-standard applications commonly used for web development and deployment. Java 2 Enterprise Edition compliance and BEA™ WebLogic® application certification combine to provide a fast, scalable, and secure configuration service.

PeopleSoft Advanced Configurator interacts within a multi-tier structure called the Lightning Architecture™ that is based on Java 2 Enterprise Edition (J2EE) technology.

The following diagrams the architecture of Advanced Configurator.



Advanced Configurator Architecture

Advanced Configurator Implementation

This section discusses:

- Implementing an integrated Configurator solution.
- Implementing a standalone Configurator solution.
- Other sources of information.

Implementing an Integrated Configurator Solution

If you intend for users to configure the product from within a PeopleSoft CRM application, you do not need to create a user interface (although it is still an option). PeopleSoft CRM pages can provide the user interface for you, requiring only that you specify the pages' data content. Optionally, you can create a custom user interface using JSP.

Step	Reference
1. Install Configurator and set it up for integration. (For integration with PeopleSoft CRM Order Capture).	<i>PeopleSoft Enterprise CRM 9 Installation Guide</i>
2. Build a model of the product(s) or service(s) using the PeopleSoft Visual Modeler.	Part 2, "Product Modeling with a Component Model," page 9 and Part 3, "Product Modeling with Compound Models," page 121
3. Deploy the solution files to the production environment using the Advanced Configurator Administration console.	Part 8, "Advanced Configurator System Administration," page 411
4. Set up the solution schema.	Part 5, "PeopleSoft CRM Order Capture Integration," page 271

Implementing a Standalone Configurator Solution

A solution that runs on PeopleSoft Advanced Configurator independent of other PeopleSoft applications requires a custom User Interface in addition to the configuration model. Advanced Configurator is also extensible, and the solution may include Java implementations of the Configurator interfaces.

Step	Reference
1. Build a model of the product(s) or service(s) using the PeopleSoft Visual Modeler.	<ul style="list-style-type: none"> • Part 2, "Product Modeling with a Component Model," page 9 • Part 3, "Product Modeling with Compound Models," page 121
2. Build a UI. Create the JavaServer Pages for the User Interface using the provided templates and, if desired, Dreamweaver extensions.	Part 6, "Building a Custom User Interface," page 295
3. Write Java extensions for any custom functions and modify the JSP pages as needed.	Part 4, "Application Extensions," page 153 Also consult the PeopleSoft Advanced Configurator API Reference Guide in your server installation root\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\calico\apidocs.
4. Deploy the solution files to the production environment using the Configurator Administration Console.	Part 8, "Advanced Configurator System Administration," page 411
5. Install the Mobile Configurator Packaging Tool and clients on the appropriate machines if you are deploying the Mobile Configurator client, then package and distribute the solution to mobile users.	<i>PeopleSoft Enterprise CRM 9 Installation Guide</i> Part 7, "Mobile Product Configuration," page 385

CHAPTER 2

Other Sources of Information

In the planning phase of your implementation, take advantage of all PeopleSoft sources of information, including the installation guides, table-loading sequences, data models, and business process maps. A complete list of these resources appears in the preface in the *PeopleSoft Enterprise CRM 9 Application Fundamentals PeopleBook*, with information about where to find the most current version of each.

See Also

PeopleSoft Enterprise Order Capture 9 PeopleBook

PART 2

Product Modeling with a Component Model

Chapter 3
Understanding Modeling

Chapter 4
Setting Up the Modeling Environment

Chapter 5
Creating Objects for the Model

Chapter 6
Creating Relationships Between Model Objects

Chapter 7
Specifying Quantities on Selection Points

CHAPTER 3

Understanding Modeling

This chapter discusses:

- Basic model concepts.
- Relationships between objects
- Relationship properties.
- Expressions in relationships.
- Default values within expressions.
- Quantities in Modeling.
- Creating parameterized explanations.
- Optimizing performance and minimizing model maintenance
- Using the sample models.
- The modeling process.
- Model Tester.
- Interfacing with third-party tools.

Basic Model Concepts

The PeopleSoft Visual Modeler is a graphical hierarchical modeling tool for designing complex configuration solutions. You use it to create a model of a product or a service that then serves as the “blueprint” for specifying actual instances of the product or service that are customized to a customer’s needs.

Advanced Configurator models represent products and services using three concepts:

- The hierarchy—a logical structure that identifies and organizes its components.
- Relationships—how the components relate to each another and interact.
- Components—a product’s parts are represented by classes, attributes, domain members, and selection points.

Visual Modeler

Advanced Configurator technology supports configuration modeling and runtime configuration processing.

Using the concepts of hierarchy, components, and relationships, Visual Modeler allows you to describe even complex products and services:

- Uses a multi-paned window to simplify model navigation and the creation of classes, attributes, and relationships.

- Uses a “no programming” paradigm.

All modeling is accomplished through drag-and-drop operations and table selections.

- Integrates easily with existing customer data.
- Provides support for team development.
- Separates the UI design from the product modeling problem.

Modelers don't need to create a UI in order to test their model logic.

Model data can be defined in the model (internal data), or obtained from a relational database.

The Configurator engine uses a compiled version of the Visual Modeler model to process user picks, ensuring a valid configuration at runtime.

In this document, a general reference to the Advanced Configurator includes both design-time and runtime components.

You can use Visual Modeler stand-alone or with the Advanced Configurator engine. The Visual Modeler interface uses common hierarchical concepts:

- A model is built from objects. An object is a functional component such as a class, attribute, relationship, or domain member.
- A class is a group or category of like things; for example, products or services.

A class defines common attributes such as color, weight, power requirements, or price.

- A class hierarchy applies a hierarchical organization to a group of classes.

Attributes are defined on a class. A new class that inherits attributes from an existing “ancestor” class is a “subclass” or “child.” The nearest ancestor class is referred to as the “parent.”

- A class member, or “domain member” is an instance of a class that assigns specific values to class attributes.
- A relationship, such as compatibility or incompatibility, can be defined between classes or class attributes, and between selection points.

Visual Modeler Objects

Visual Modeler employs standard object-oriented principles for class and inheritance. A model includes an automatically generated Root class, and any classes, subclasses, domain members, and relationships that you define. Inheritance moves from left to right, such that child classes and domain members inherit attributes from parent classes. Hierarchical modeling makes a large configuration task more approachable. Classes can be grouped in meaningful ways, and inheritance capabilities can greatly reduce the work of explicit value assignment.

The compiled model provides basic configuration functionality. To visually organize the display of controls at run time, use DreamWeaver with the Advanced Configurator Extensions. Using DreamWeaver reduces the need to hand-code JSP pages. Model functionality can be further extended with custom Java code that manipulates the controls and options displayed at run time.

Object Properties and Attributes

All Visual Modeler objects (classes, domain members, or relationships) have properties, also referred to as system properties. Classes can have both properties and attributes.

- Properties are part of the object definition and cannot be removed.

For example, classes, domain members, and relationships all have a Name property. You supply a string to define this property. A property setting is specific to an object.

- Attributes can be thought of as optional user-defined extensions to a class description.

Attributes can be changed or deleted at any time. Attributes are inherited by subclasses. Domain members assign values to attributes, creating an instance of a class.

Classes and Class Attributes

A class is a group of related objects—items or characteristics a user can order, specify, or require.

- The Root class is a special class defined in every model.

It behaves like any other class, except it can't be deleted and can't have domain members.

- Any number of attributes can be added to a class.

Attributes can be of type Boolean, Float, Int, String, and Date. User-defined class attributes appear on all subclasses and subsequent domain members.

- Class attributes can be assigned default values only in the class on which they are defined (class attribute values cannot be altered in subclasses).

An attribute value can be assigned on a domain member instance as well.

- The Internal flag is a class property that determines whether internal domain members will be used, or if domain members will be obtained from a database. A list of class properties follows.
- When external domain members are used, an attribute must exist for each column pulled from the database. This includes the Name. The name defined when the class was created exists internally, so it is not available if the Internal attribute is set to False. (Attribute names should begin with a letter. Attribute names starting with “_” and “\$” are reserved for use by the Advanced Configurator system.)

Visual Modeler supplies the following default properties for each class:

Name (String)	The class name specified at creation.
File Name (String)	The file name specified at creation. Although the class name you see in the Visual Modeler can be changed interactively, the corresponding file name cannot be changed.
Internal (Boolean)	Default is <i>True</i> , implying that domain members are internal (defined within the model). If this property is set to <i>False</i> , the SQL Query dialog will be displayed. A model can combine both internal and external domain member data, although not within a single class.
SQL Query	Only displayed if Internal is set to <i>False</i> . Click on the Edit button to raise the Primary Table dialog. This dialog allows you to specify the data source, the table used, and the columns accessed.

You must consider the effect of inheritance principles as you build a model. Click anywhere in the Model Structure View to ensure focus on the modeling area.

See [Chapter 5, “Creating Objects for the Model,” Creating a Class, page 69](#).

Domain Members

A domain member is an instance of a class that describes a particular item, service, or decision. Domain members assign values to class attributes. A class can have either internal or external domain members, but the Internal flag determines the source of the domain members used in the model.

Internal Domain Members

If a domain member is internal, values are assigned as part of the domain member definition. An internal domain member has only one property: its name. A domain member can instantiate attributes from the parent class.

See [Chapter 5, “Creating Objects for the Model,” Creating Internal Domain Members, page 73](#).

External Domain Members

External domain members populate the model based on the SQL Query defined for the class. Values extracted from a database are assigned to a corresponding class attribute. External domain members can be internalized using the menu command Project, Internalize Model. This tool is primarily for use when you need to package a model with its data, such as for transfer and setup at an off-line location for support and testing.

See [Chapter 5, “Creating Objects for the Model,” Setting Up Binding for External Domain Members, page 75](#).

Selection Points

A selection point is an object in the model that will be exposed at run time. A class or a class attribute can become a selection point (sometimes referred to as a decision point). Only selection points shown in the model structure view are available for display at run time.

In addition, a selection point:

- May or may not be visible and selectable at run time.
- May or may not be required to satisfy the runtime completeness check.

By default, a selection point is created for every leaf class that participates in a relationship.

It also participates in all relationships that refer to the original parent class. If a relationship is made directly between a selection point and another object, that relationship is confined to the selection point.

Relationships Between Objects

The relationships you can define between objects in Visual Modeler are:

- Compatibility constraints
- Requirement constraints
- Dynamic defaults
- Resource constraints
- Summation
- Elimination
- Comparison

Compatibility

A non-directional compatibility constraint explicitly identifies all *valid* combinations and eliminates all other possibilities. Consider the example of an eyeglass product in which Sport frames are compatible with plasticShatterproof lenses. Thus, a pick on either Sport or plasticShatterproof eliminates all other choices. The single remaining choice will be computer-selected if the control is not optional.

Note. Any options not explicitly marked as compatible are assumed to be incompatible. There is no neutral state in a compatibility constraint.

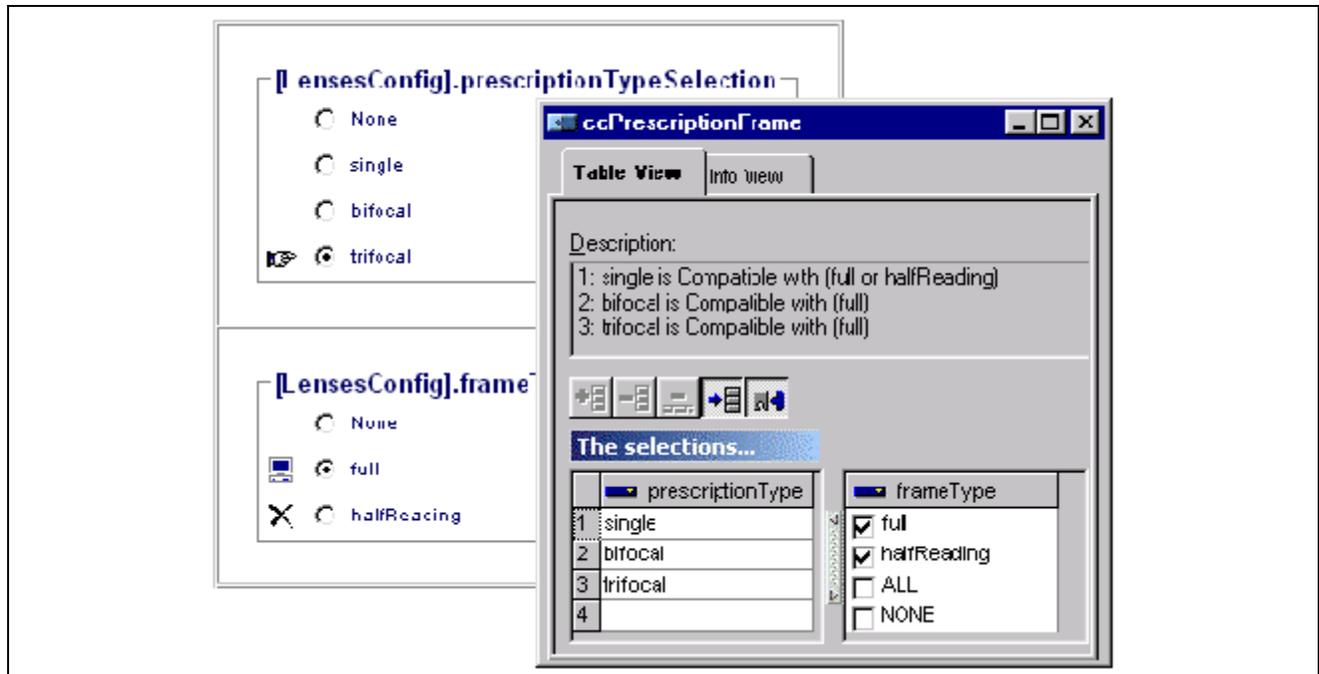
Directional Compatibility

A directional compatibility constraint has a left-hand side (LHS) and a right-hand side (RHS). It identifies compatible combinations, and eliminates all other selections. In the relationship editor, the constraint is expressed in a table with a directional bar separating the LHS arguments from the RHS arguments. The bar indicates that the combined LHS selections are compatible with one of the RHS items; the RHS items are considered separate and unrelated.

- At run time, the constraint can't eliminate all incompatible items until a pick or a computer selection is made on all but one of the controls. In other words, given N columns in the constraint, the constraint is not "bound" until a user or computer pick is made on $N-1$ controls. Then the remaining pick is calculated and all incompatible items are eliminated.
- If a RHS object is optional, elimination will occur but a computer selection cannot be made. If the RHS object is required and there is only one valid selection on the RHS, it will be computer-selected.
- Do not use directional compatibility if any RHS object is multi-select. A directional compatibility cannot eliminate items from a multi-select control. Since nothing is eliminated, nothing is computer-selected so the constraint has no effect. If a RHS argument is set to multi-select, you will see the following message at compile time:

```
multi-select decision point <nameSelection> in RHS of
directional compatibility=>
  constraint <constraintName>
will be ignored
```

The following figure shows an example of relationship dialog settings and the HTML result of a directional compatibility:



Directional compatibility

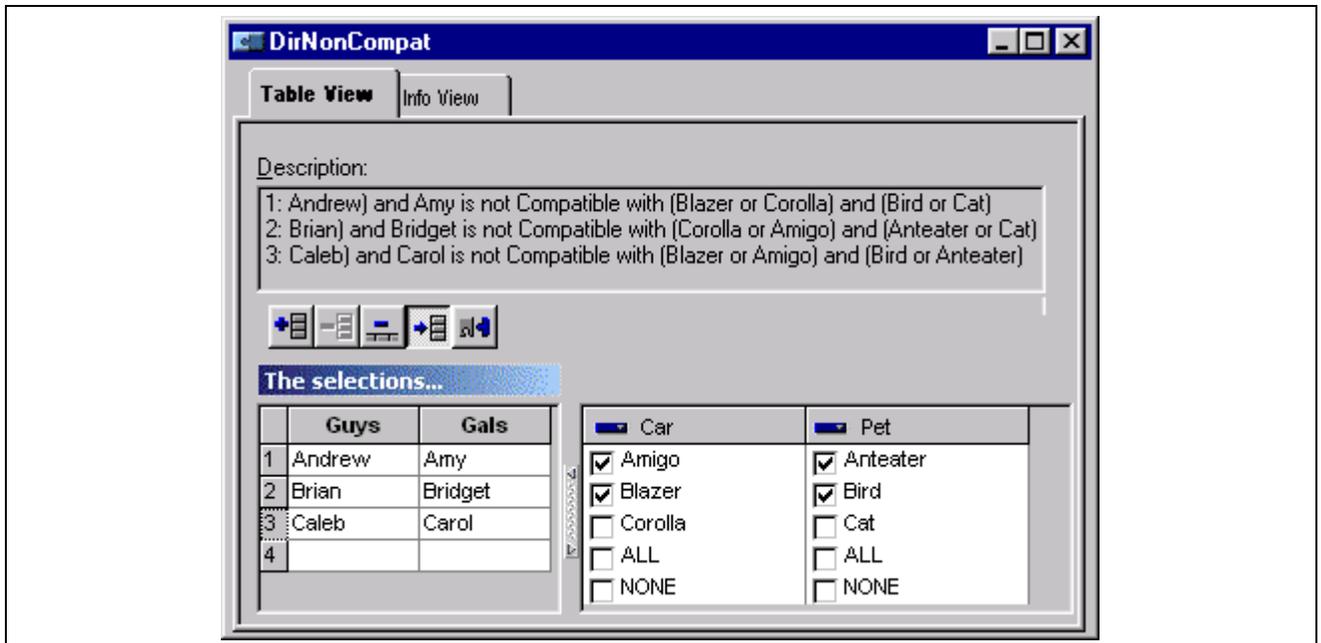
Non-Compatibility

A non-compatible constraint enumerates all invalid combinations. If any member of an invalid combination is selected, all members are eliminated. This is true for both single- and multiple-select controls.

- If a control is not optional and only one option remains, that option will be computer-selected.
- Generally, in a non-compatibility constraint, when there are more than two arguments in a constraint (that is, more than two columns in a relationship editor table), the full set of eliminations will not appear unless there is a user pick or a computer-selection for all but one selection. In other words, given N columns in the constraint, the constraint is not “bound” until a user or computer pick is made on $N-1$ controls. Then the remaining pick is calculated and all incompatible items are eliminated. In some cases, however, it’s possible for fewer than $N-1$ selections to cause eliminations to appear. For example, in a three-column, non-compatibility constraint, if a value from the first column along with a value from the second column are incompatible with every value from the third remaining column (and that column is a required selection), then selecting the value from the first column will eliminate the corresponding value from the second column (since the additional selection of the value from the second column would eliminate all selections for the third column).

Directional Non-Compatibility

A directional, non-compatible constraint evaluates a pick and then eliminates all the incompatible items. In the relationship editor, the constraint is expressed in a table with a directional bar separating the LHS arguments from the RHS arguments. Row by row, the combined LHS selections are incompatible with one of the RHS items. The RHS items are separate and unrelated as shown in the figure.



Relationship dialog for a non-directional compatibility

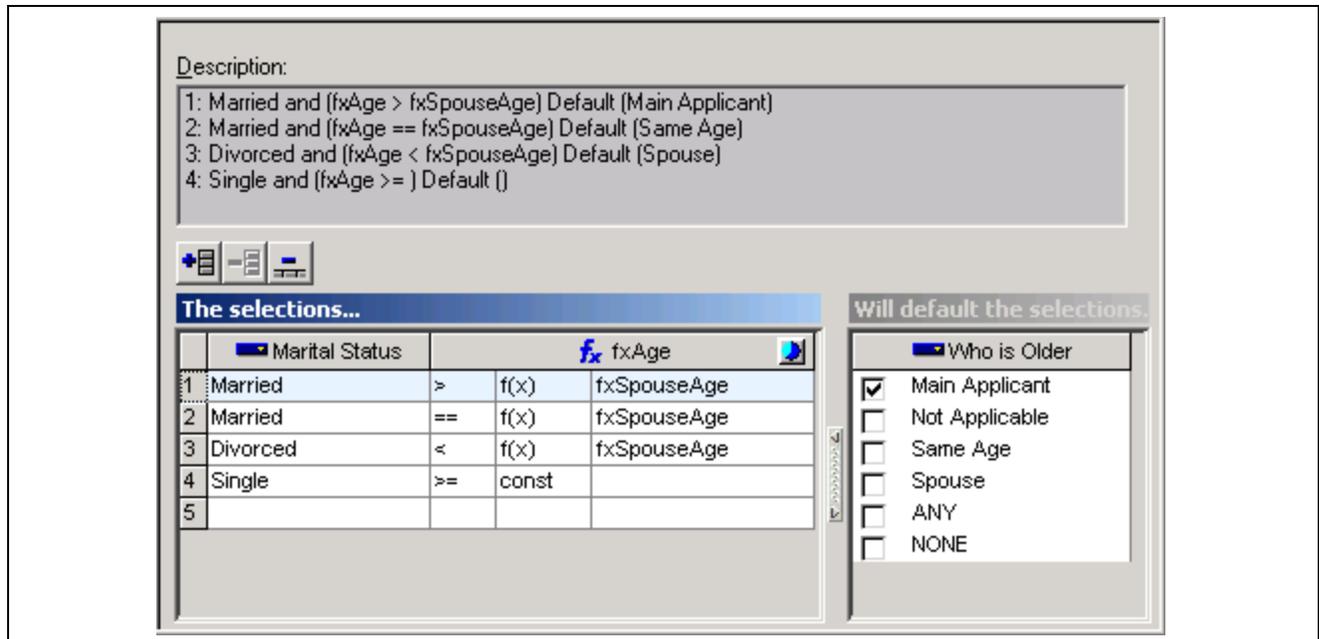
Because RHS items are unrelated, a directional non-compatibility is equivalent to many constraints. For example, row 3 equates to:

- Caleb and Carol are incompatible with Amigo.
- Caleb and Carol are incompatible with Blazer.
- Caleb and Carol are incompatible with Anteater.
- Caleb and Carol are incompatible with Bird.

Expressions in the Left-Hand Side of the Relationship

Using expressions in the LHS of the relationship lets you to constrain against values that aren't known until run time. Since expressions name variables whose values are input at run time through user- or database input, by using expressions you can describe a condition or set of conditions that result in a desired default selection or quantity on a selection, or both.

For example, a model for financial services product offers differing plans based on the customer's marital status and the spouse's age. You can use an expressions on the LHS for a requirement constraint to determines whether the customer or their spouse is older and thus which partner is considered the principal applicant. The selection "Main applicant" is computer-selected if the customer is older than the spouse.



Requirement constraint: If “Married” is selected, and the result of the expression fxAge is greater than fxSpouseAge, then “Main Applicant” is selected.

The example illustrates the two capabilities that using expressions in the LHS of a relationship allow you:

- Eliminate items (in the case of a compatibility constraint) and require items (in the case of a requirement constraint),
- Define a range of values that default a selection.
- Describe conditions that default a selection that can’t be known at design time, especially in the case of external data.

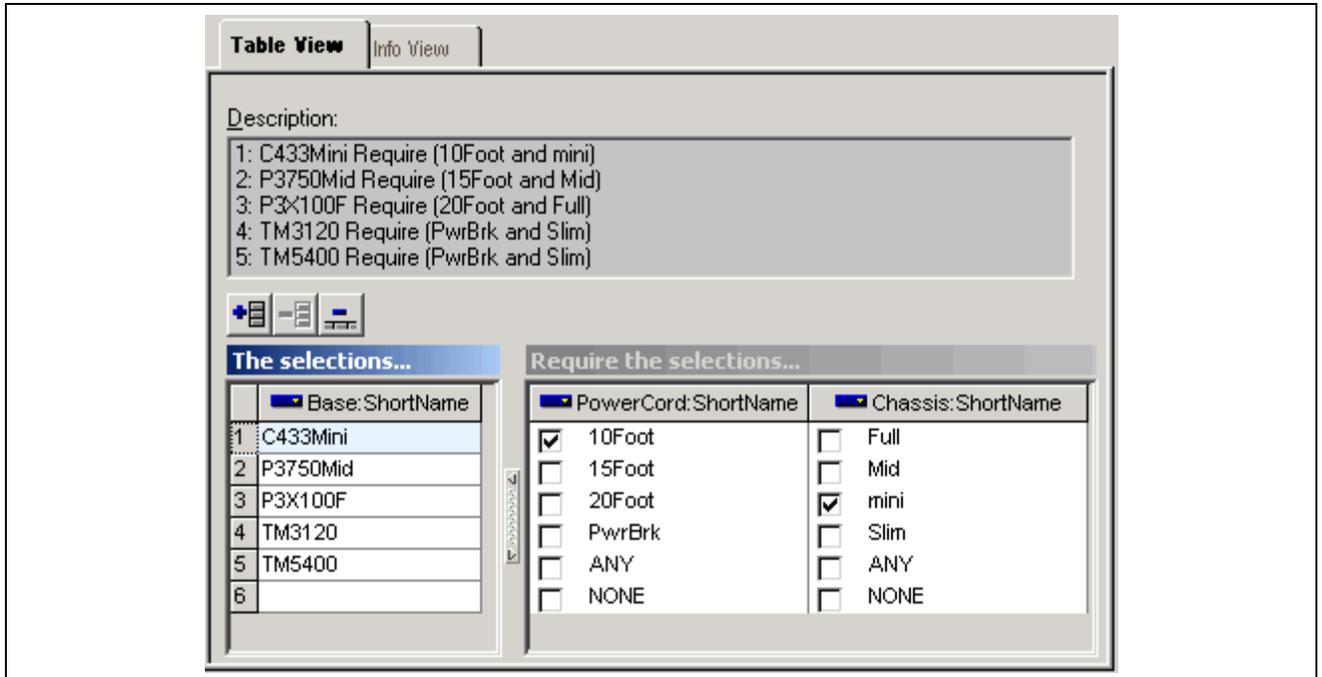
In defining a range of values, you can describe the lower limit, the upper limit, or both. In addition, you can use more than one expression to define the default conditions.

Requirement Constraint

A requirement constraint makes a computer selection on items that meet the LHS criteria. The figure below shows a simple Requirement, where selecting “photogray” causes “glass” to be computer-selected.

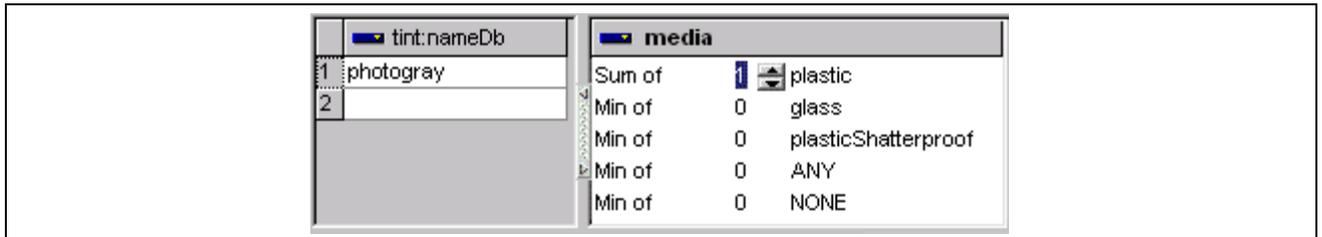
Requirement constraints have these characteristics:

- Requirement constraint behavior is unaffected by optional/required status on a selection point.
- If an RHS control is single-select, the selection is made and the rest of the items are eliminated. In the multi-select case, the computer selections on the RHS are made, but no eliminations occur.
- Given the constraint shown in the figure, if the RHS item “glass” is eliminated (by a different constraint), then the LHS item “photogray” will also be eliminated.
- Externs can be used on the LHS and the RHS of a requirement constraint.



Requirement constraint

When the Requirement is written on a Selection Point, you can define default quantities on the selections. If the Selection Point has Quantity property set to True, settings appear in the column.



Requirement constraint with quantity settings

See [Chapter 6, “Creating Relationships Between Model Objects,” Editing Requirement Constraints, page 99.](#)

Dynamic Default

As mentioned earlier, a Dynamic Default is not a true constraint because it is not considered in the model verification process. There are some similar traits, however. A Dynamic Default is directional. When the LHS criteria is met, a computer-select occurs on the item on the RHS, provided it is available and selecting it will not cause a violation. The RHS can have multiple arguments (multiple columns in the relationship table), in which case a default pick can be made on each control.

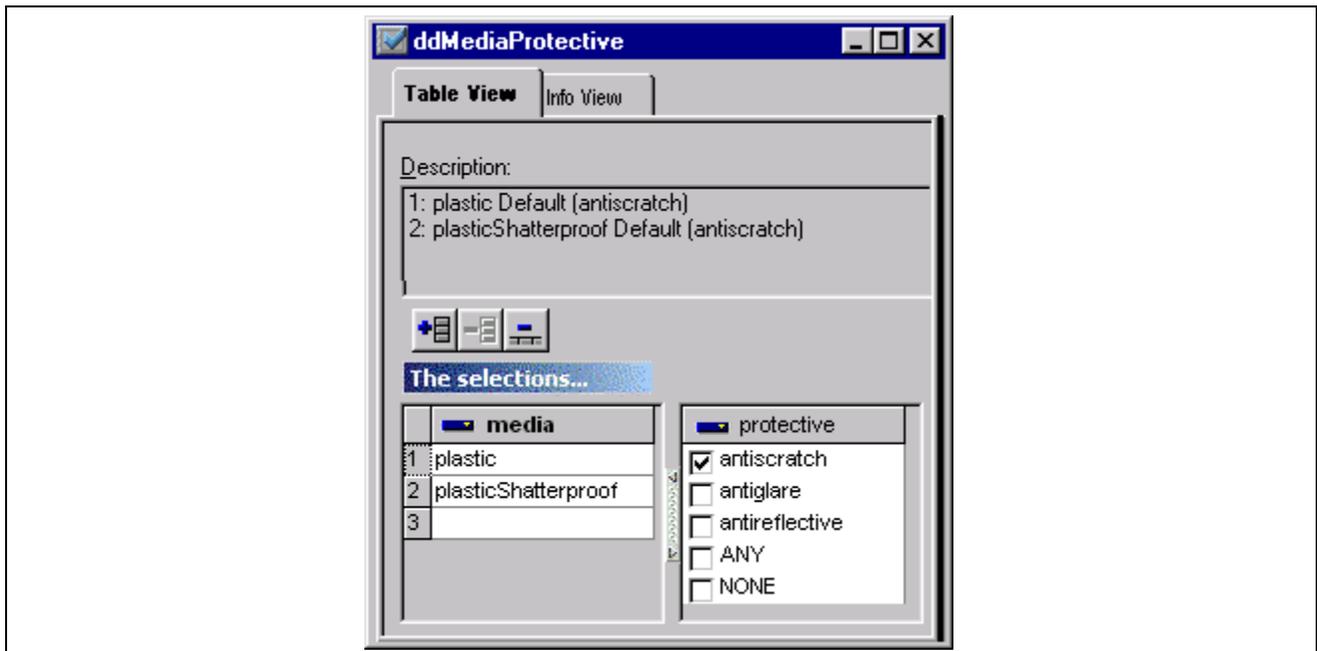
See [Chapter 6, “Creating Relationships Between Model Objects,” Editing Dynamic Defaults, page 100.](#)

A dynamic default has these characteristics:

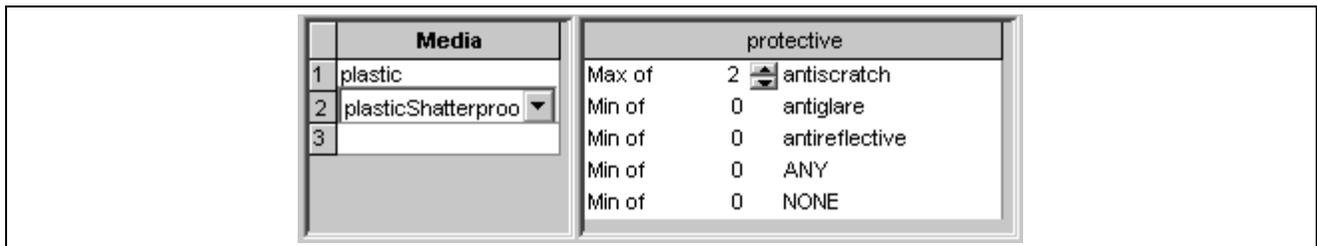
- If a default item has been eliminated, nothing is done.
- Once the default pick exists, any constraint reference to the item can eliminate it. There is no violation when a Dynamic Default is superseded.

The following figure shows a simple default with two rows. Each LHS argument has a different pick on the RHS. “Plastic” picks “anti-scratch,” and “plasticShatterproof” picks “anti-reflective.”

- As with Requirement constraints, when the default is written on a selection point, you can define quantities for the selection. If the selection point flag Quantity is set to True, settings appear in the column.



Dynamic default relationship



Dynamic default with quantity settings

Resource Constraint

A Resource constraint evaluates numeric attribute values. A constraint designates one or more provider attributes and one or more consumer attributes. If the sum of the consumers exceeds the sum of the providers, a conflict occurs and an explanation, if defined, is displayed at the page level. This behavior can also be achieved with a numeric Comparison.

See [Chapter 6, “Creating Relationships Between Model Objects,” Editing Resource Constraints, page 101.](#)

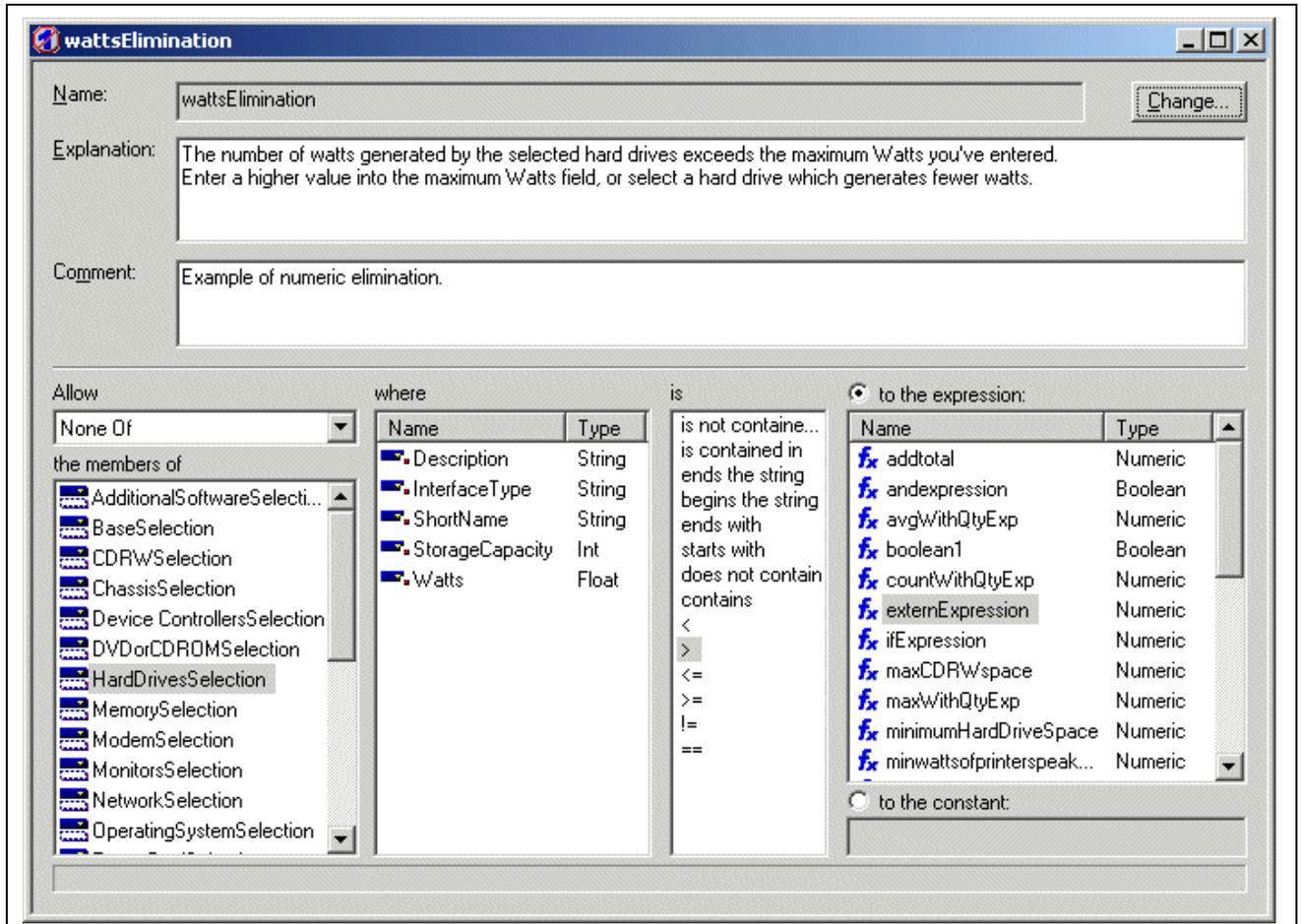
Summation

The summation relationship adds the value of numeric attributes. The sum can be displayed at run time. The summation relationship does not affect model verification.

See [Chapter 6, “Creating Relationships Between Model Objects,” Editing Summation Relationships, page 102.](#)

Elimination

An elimination compares a specific attribute value on a selection point with the value of a numeric, boolean, string, or date expression. The following figure shows an elimination where domain members of the selection point “HardDriveSelection” are eliminated if the value of the selected hard drive’s Watts attribute exceeds the value passed in by externExpression. An explanation appears at the page level when the constraint is violated.



Elimination of numeric types

See [Chapter 6, “Creating Relationships Between Model Objects,” Editing Elimination Constraints, page 103.](#)

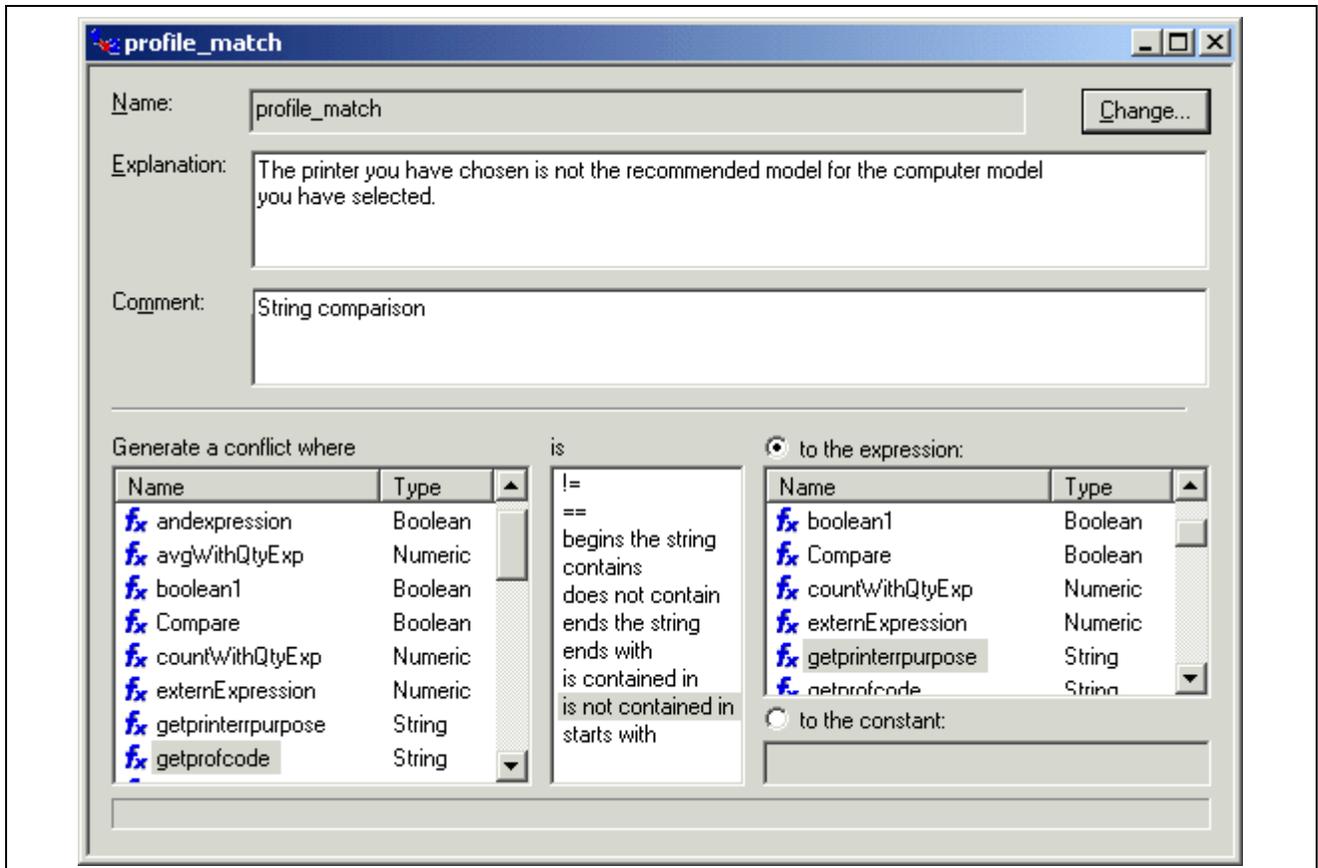
Comparison

Advanced Configurator supports comparison relationships for the four data types string, numeric, boolean, and date. If the comparison is False, a conflict occurs, and an explanation, if defined, appears at the page level. Because comparisons operate on expressions, it does not pick or eliminate domain members.

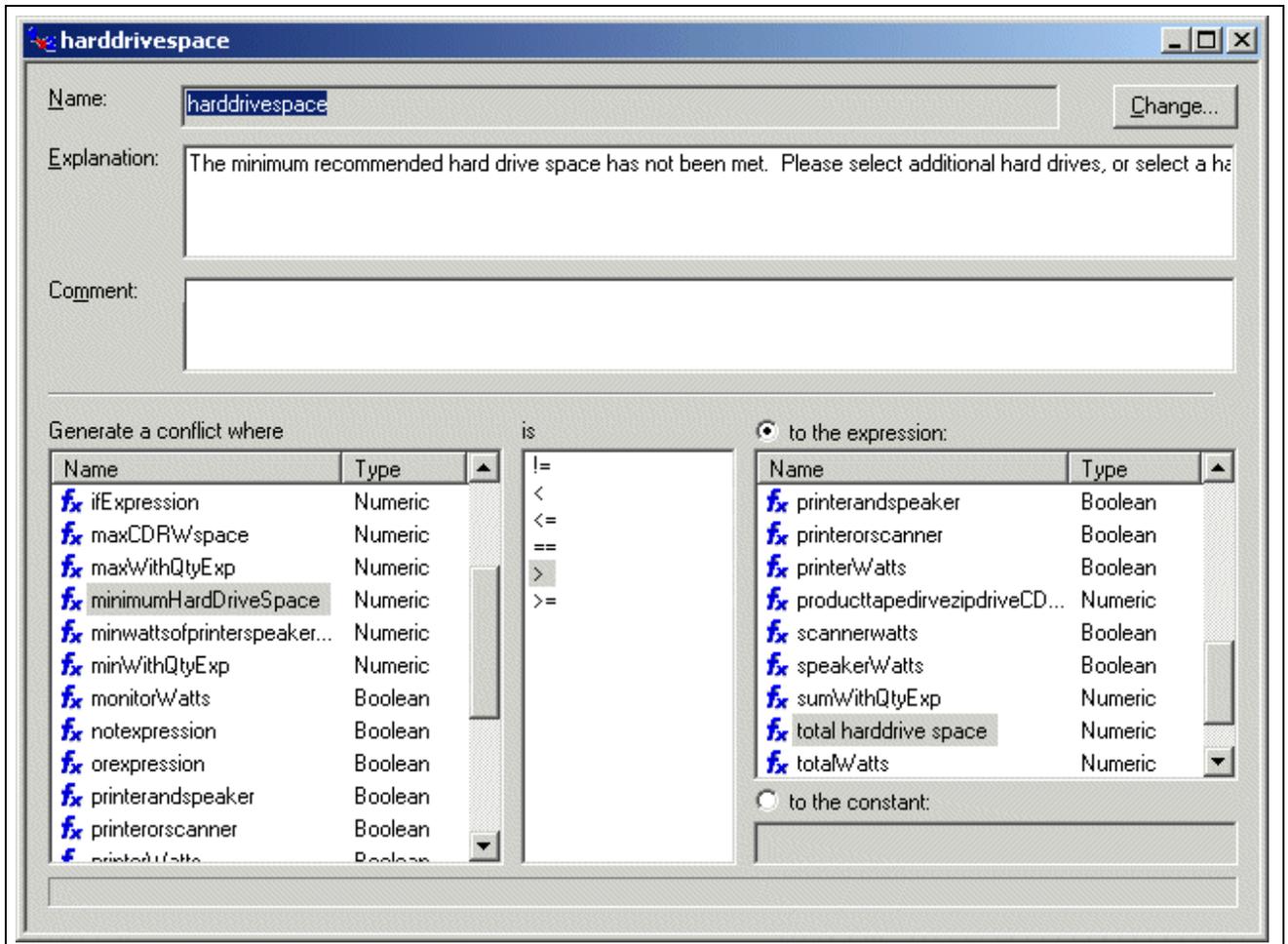
A numeric comparison compares the value of one numeric expression with the value of another; a boolean comparison compares boolean values; a string comparison, strings; and a date comparison, dates. Or, you can compare the value of an expression with a constant rather than an expression with the “to the constant” option.

The figures illustrate examples of a string, numeric, and boolean comparison.

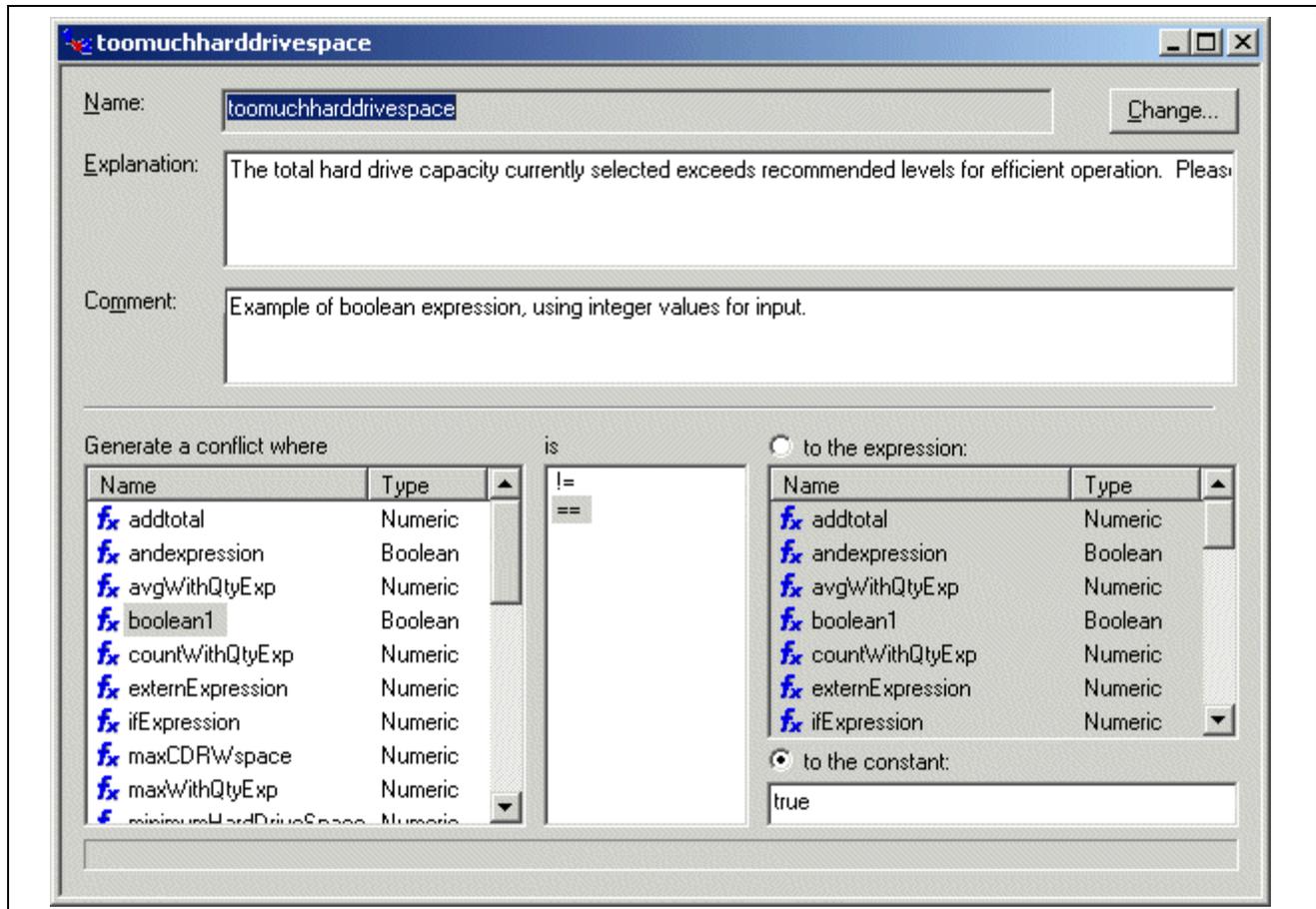
See [Chapter 6, “Creating Relationships Between Model Objects,” Editing Comparison Constraints, page 104.](#)



String comparison relationship



Numeric comparison relationship



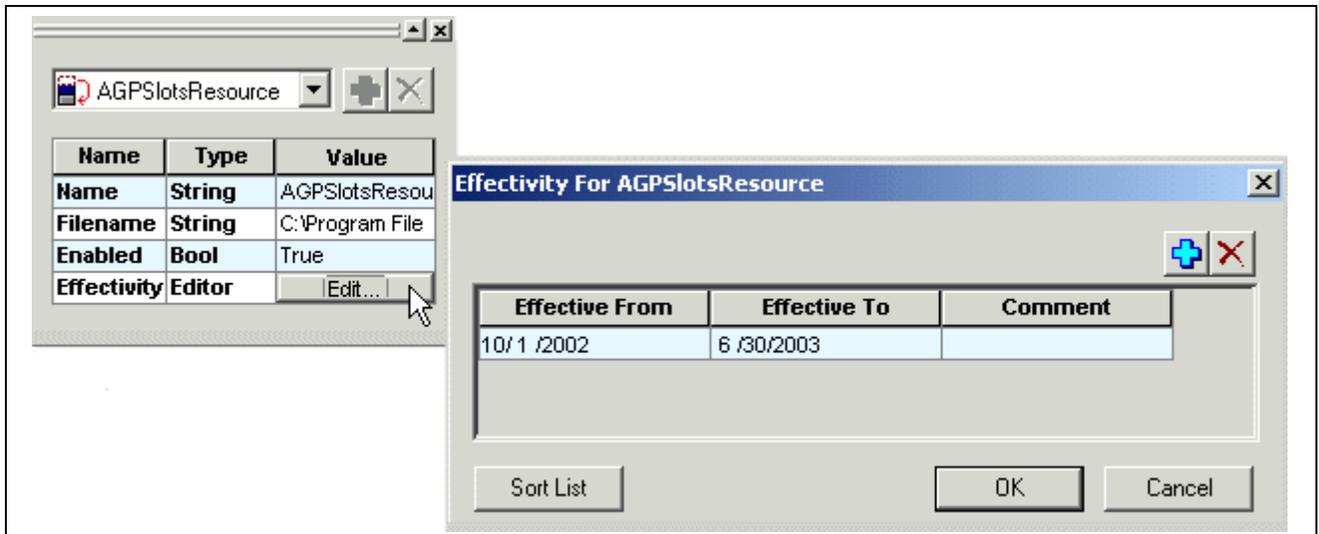
Boolean comparison relationship

Effectivity Dates

Relationships and comparisons have optional date of effectivity ranges to indicate when they are to be considered active and thus used in a configuration session. With effectivity dates, you can define constraints and defaults that apply only during special sale periods, holiday periods, or other times when the valid or suggested configuration is slightly different.

In contrast, expressions and summations do not have effectivity dates because their results may be used in other, active relationships and expressions, and may be displayed on the UI. You can set effectivity dates on compatibility and incompatibility constraints, dynamic defaults, resource and requirement constraints, comparisons, and eliminations.

Effectivity dates are set in the relationship's table editor using the Effectivity dialog. Click the Edit button to open the dialog.



Effectivity in the property editor and in its dialog

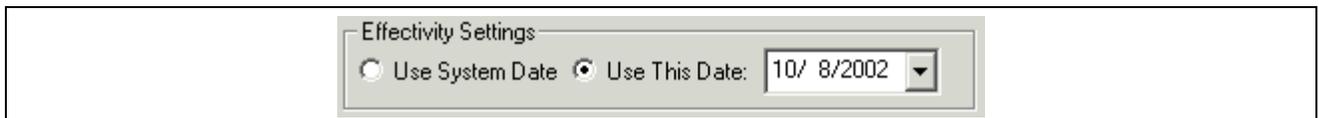
You can enter multiple date ranges by clicking the Add button.

At run time, the server's system date or a specific solve date is compared to the date ranges specified on each of the relationships in the model. If it falls within any date range specified on a relationship, then that relationship will be included in the configuration session. As soon as the Configurator engine detects at least one valid range on a relationship, it will cease further comparison against any remaining date range rows and the relationship will be enabled. If the date entered at run time does not fall within any date range specified on a relationship, then that relationship is disabled. Relationships that have no date ranges specified are considered to always be enabled.

Active relationships will be executed and propagated, and their results returned to the end-user UI. Disabled relationships will not be executed or participate in propagation, nor will they interact in any way with the configuration.

Any date can be entered at run time, however, if no date value is entered, the current date will be used.

To facilitate model testing, Visual Modeler allows you to specify a date to use as the solve date, so that when the model is run, the engine will use that date rather than the system date or a specified date. The model test solve date is set in the Projects Settings dialog. In addition, the Model Tester itself lets you change the solve date without recompiling the model.



Setting the compile effectivity date for testing in the Projects Settings dialog

Expressions in Relationships

Note that every expression you create appears at the bottom of the Model Tester with its current value. This will help you determine what is happening when you test your model.

Expressions will evaluate without selections or user entries. If a referenced object does not have a user selection or user-entered value, the default value for the object will be used. If you don't want the expression to evaluate, you can use the "bnd()" function around the participating objects.

Note. The model will still verify as true when expression values are missing or incomplete. Because an unbound constraint is never evaluated, no conflict is generated. Use the “bnd()” function with any objects that you want to remain optional.

A pick can be a user pick, *None* on a single-select, default picks, or computer-selections. This is a limiting factor if you want to use multi-selects or if you want items to be optional. This example shows a workaround for the multi-select case, where an extra domain member *No Thanks* has been added and is selected by default.

This figure also demonstrates explanations for relationships that use expressions. The total exceeds the budget amount specified, so a message is displayed at the top and the configuration evaluates to false.

Violations
 You are over budget! This is only a warning.

<p>Service - \$10 certificates</p> <ul style="list-style-type: none"> <input type="checkbox"/> Book Barn <input type="text" value="3.0"/> <input type="checkbox"/> Bright Books <input type="text" value="0.0"/> <input type="checkbox"/> CD Station <input type="text" value="0.0"/> <input type="checkbox"/> Movie Passes <input type="text" value="4.0"/> <input type="checkbox"/> No Thanks <input type="text" value="0.0"/> 	<p>Service - \$25 certificates</p> <ul style="list-style-type: none"> <input type="radio"/> None <input type="text" value="1.0"/> <input type="radio"/> Car Wash <input type="radio"/> Manicures by Mona <input type="radio"/> Massage by Morris <input checked="" type="radio"/> No Thanks
<p>Entertainment</p> <ul style="list-style-type: none"> <input type="checkbox"/> Online Game - \$60 per year <input type="text" value="0.0"/> <input type="checkbox"/> PC Game Monthly - \$150 per year <input type="text" value="0.0"/> <input checked="" type="checkbox"/> Adult Fun Season Pass - \$250 <input type="text" value="0.0"/> <input type="checkbox"/> Child Fun Season Pass - \$199 <input type="text" value="0.0"/> <input checked="" type="checkbox"/> No Thanks <input type="text" value="1.0"/> 	<p>Hobbies</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Cigar of the Month - \$85 per year <input type="text" value="0.0"/> <input checked="" type="checkbox"/> Beer of the Month - \$75 per year <input type="text" value="0.0"/> <input type="checkbox"/> Fantasy Football 2001 - \$25 per year <input type="text" value="0.0"/> <input type="checkbox"/> Bulb of the Month - \$75 per year <input type="text" value="0.0"/> <input checked="" type="checkbox"/> No Thanks <input type="text" value="1.0"/>
<p>budget: <input type="text" value="50"/></p>	<p>age: <input type="text" value="11"/></p>
<p>AdultEvalNumeric: 12.0</p>	<p>BudgetNumeric: 50.0</p>
<p>Total: 70.0</p>	<p>RecipientAgeNumeric: 11.0</p>

Expression behavior and display at run time

Note. Advanced Configurator doesn't support date constants in expressions. Rather than using a constant, use the function `date`, `toDate`, or `intToDate` to generate the date.

For example, instead of

```
dateToInt (2002-10-24)
```

use one of these:

```
dateToInt (intToDate (20021024))
```

```
dateToInt (toDate ("2002-10-24"))
```

```
dateToInt (date (2002, 10, 24))
```

Relationship Explanations

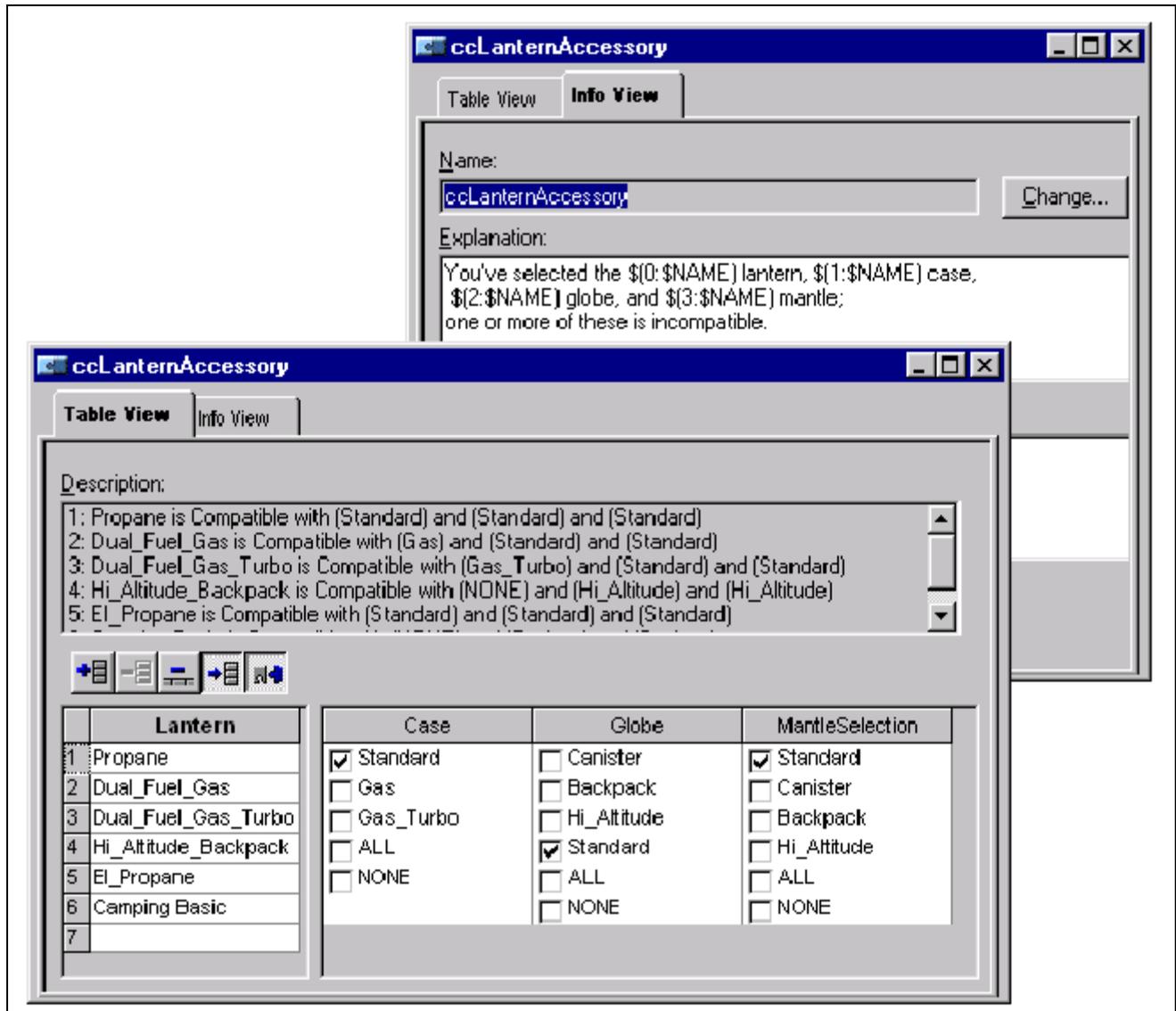
The relationship editors for constraints have a field named Explanations. For Compatibility and Requirement constraints, it appears on the Info View tab. For all others it is at the top of the form.

The Explanations field allows you to specify a message for display at run time. The explanation will only be displayed if the constraint is violated. (Text entered in the Explanations field is passed to the “Why Help” control if one is implemented for the page.)

- If the constraint is a Compatibility or Requirement constraint, the explanation will appear on all control(s) participating in the constraint and also at the page level.
- If the constraint is a Resource constraint, Elimination, or Comparison, the message appears at the page level only. In the Model Tester, this is at the top of the model.
- A parameter can be used to display the name of the pick as part of the explanation. The parameter format is `$(N:$NAME)`. You can also show other values besides the name by replacing `$NAME` with an attribute value or one of a list of provided parameters. If the domain members are external, you must refer to the database column name instead of an attribute name.

`N` is a number corresponding to a column in the relationship table. The left-most column is 0 and the number increments as you count to the right.

See [Chapter 3, “Understanding Modeling.” Creating Parameterized Explanations, page 44.](#)



Explanation field with parameters

Relationship Properties

Depending on their type, relationships can have these properties:

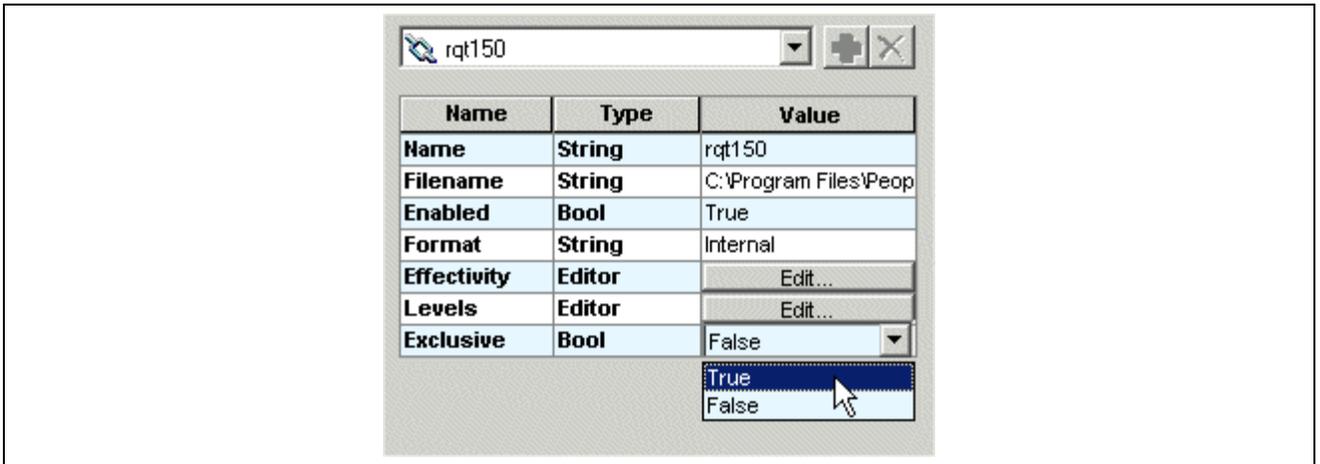
- Enabled
- Exclusive
- Format
- Effectivity Date
- Levels

Enabled

All relationships have the Enabled property. When Enabled is True (the default) the relationship is included in the model at compile time. If Enabled is False, the relationship is ignored. This setting is useful for testing relationships.

Exclusive

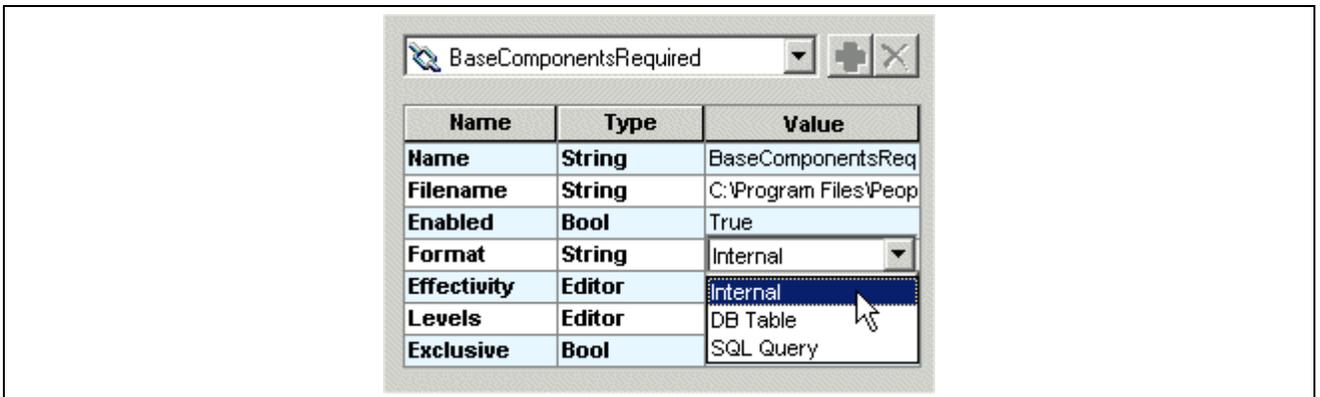
The Requirement constraint has a special field named Exclusive. Setting Exclusive to True means that a control will be reserved for the RHS element in a Requirement constraint. You must define a separate constraint for each selection point that requires a reserved RHS selection. At compile time, the selection point for the RHS argument will be “cloned” for each constraint that requires it. In this manner, a selection on the RHS can make a pick on the reserved control without eliminating that option for other constraints.



Requirement constraint exclusive property

Format

Constraints can be stored in three formats. The format can be selected as shown in the figure.



Possible format options

Internal

In the Internal format, the constraint is stored in the .cms file specified when it was created.

SQL Query

Structured Query Language (SQL) query constraints offer an alternative to constraint relationships built and maintained within the model.

For example, the standard approach to defining a compatibility constraint between a Chassis and a Drive type is to manually enter and match the Chassis

types with their compatible (or incompatible) Drive types. If compatibilities change, the relationship must be edited.

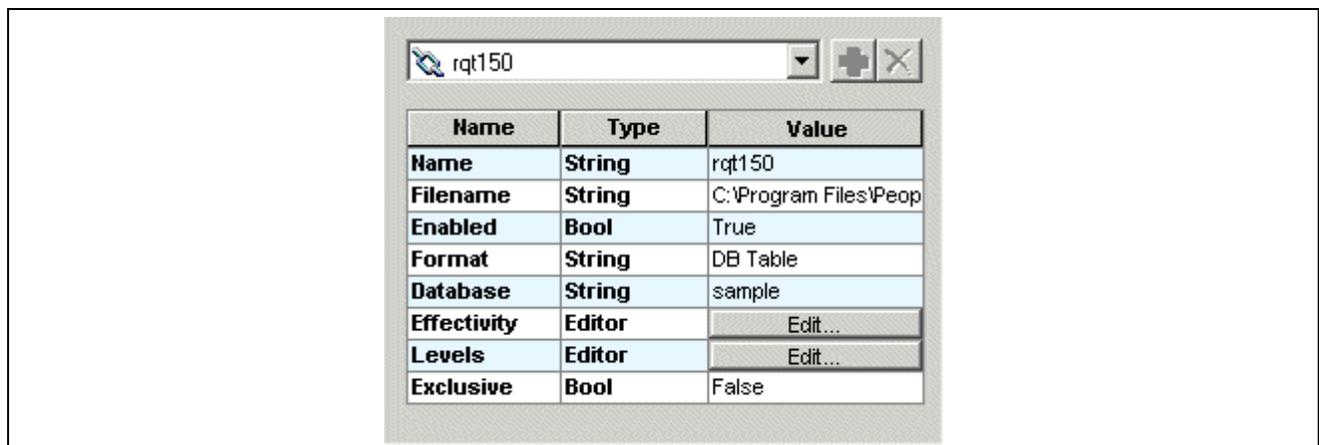
This approach, although easy to use, can prove difficult to maintain if relationships change often.

For example, if a manufacturing change causes a formerly incompatible Desktop Chassis to be compatible with the 40G Drive, you would need to edit the constraint to reflect this new relationship. For products that change often, you can use the SQL query feature, which removes the constraint definition from the model altogether and places it in a database.

See [Chapter 6, “Creating Relationships Between Model Objects,” Creating Relationships Outside the Model with SQL Queries, page 105.](#)

DB Table

When you select the DB Table option, the constraint information is saved to a table. The DB Table option can be used only for a constraint that operates on external data only; it will not work if any of the participating classes are using internally defined data. By default, a table with the same name as the constraint is stored in the current database. An additional Database row appears in the properties editor. You can specify a different database provided you have the proper ODBC driver set up for it.



Requirement constraint exclusive property

Effectivity Date

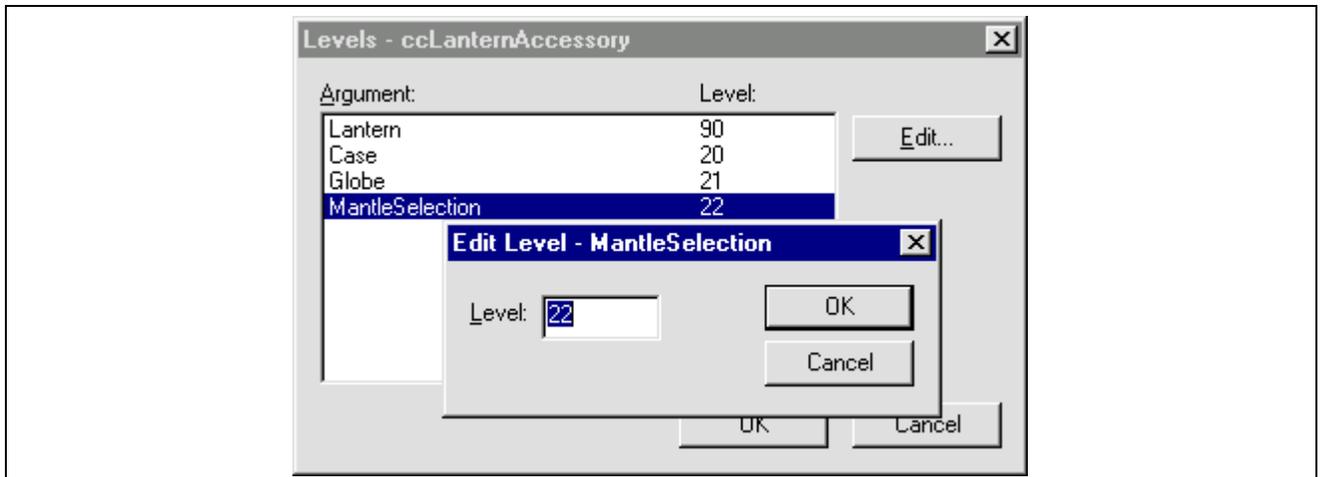
Relationships and comparisons can be set to participate in a configuration session only during one or more specified time periods. The Effectivity property allows you to enter one or more “From” and “To” date ranges. If the configuration session’s “solve date,” taken from the end-user’s system, falls within one of the specified ranges, the relationship or comparison is made available for the session.

By default, relationships and comparisons have a single effective date with the range *1/1/1900 to 12/31/2099*.

Levels

A constraint can have an arbitrary level number assigned to each argument. The level number is only returned when an item is eliminated, including when it is in violation. JSP page designers can use the level number to decide when to suppress display of certain selections.

To set a level, click the its Edit button in the Relationship properties editor to raise the Levels dialog. Select an argument, then click the dialog Edit button to change the level.



Levels dialog

In the Model Tester, if the Show Elimination Level option is checked, the lowest level number for an eliminated item will be displayed. If no elimination level was defined for a selection, the Model Tester automatically returns 1.

Default Values Within Expressions

At run time, expressions are not evaluated until all the arguments used in the expression are bound. For selection points, binding occurs when a selection has been made; for an extern, when a value has been applied. However, preferred behavior is that expressions are evaluated all of the time unless there is a specific reason to delay evaluation until all arguments are bound.

In order to enable testing when there are unbound arguments, expressions use the defined default value for unbound arguments.

Default values allow an expression to evaluate regardless of whether all of the arguments are bound. However, if you don't want the expression to evaluate unless particular arguments are bound, use the `bnd()` function within the expression. The `bnd()` function returns the value of the first bound argument or a "not bound" status if none of the arguments are bound. If `bnd()` is used within an expression and it returns the "not bound" status, the expression will not evaluate.

Example: The following expression always evaluates and returns a result even if the selection points don't yet have selections:

```
SP1:attr1 + SP2:attr2
```

To cause the expression not to evaluate unless arguments are bound, use the `bnd()` function within it:

```
SP1:attr1 + bnd(SP2:attr2)
```

You can also use the `bnd()` function in cases where a different, non-bound value is required:

```
SP1:attr1 / bnd(SP2:attr2, 1)
```

In this case, the `bnd()` function is used to return the value "1" for the numeric attribute of SP2, if SP2 is not yet bound.

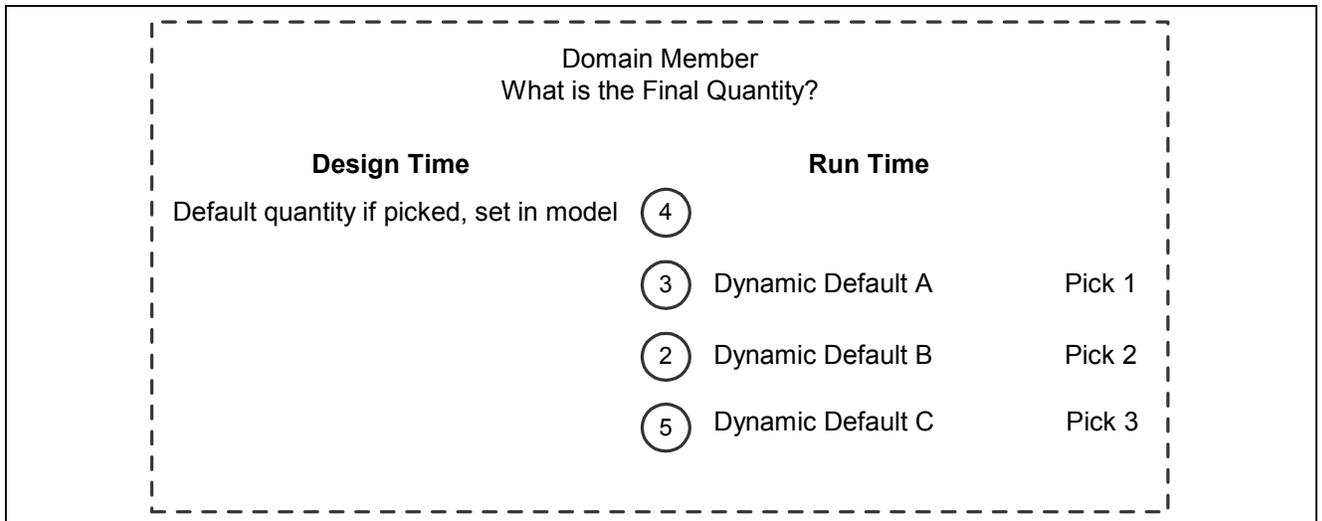
Quantities in Modeling

In a model, you can define quantities on a selection point and quantities for a selected domain member that allow you to implement a wide variety of quantity-dependent business logic.

Advanced Configurator quantity functionality lets you:

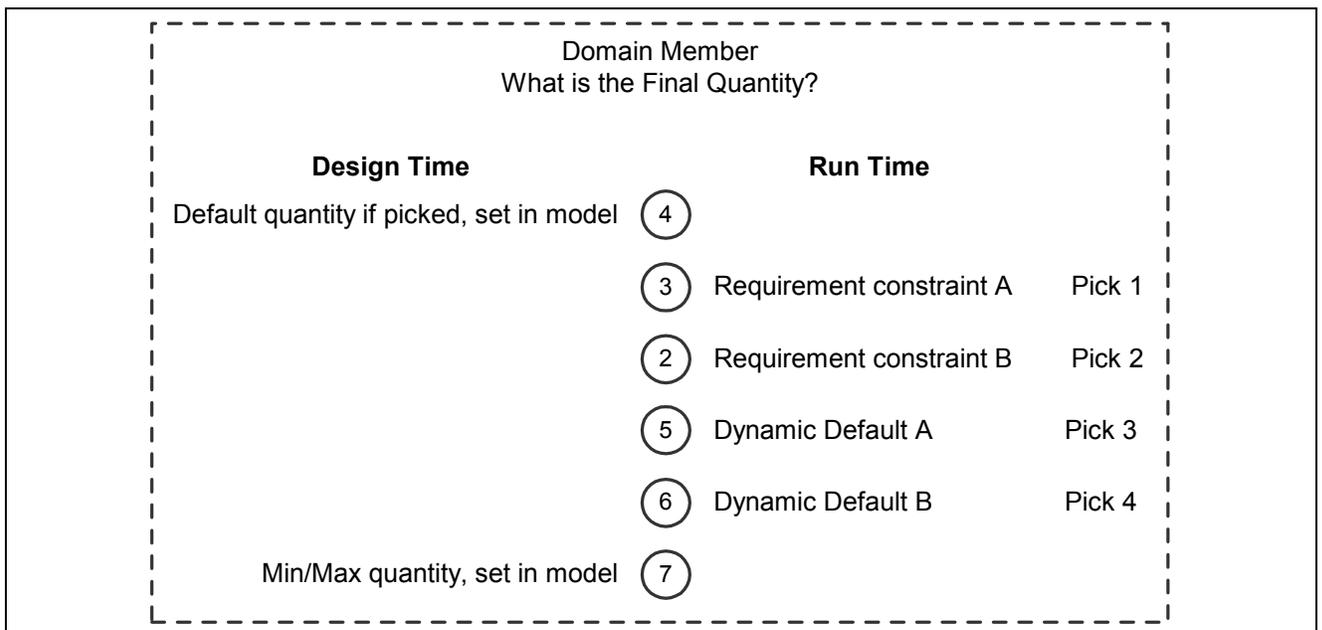
- Control how many different items on a control can be selected (single- and multi-select). The user or computer can select either ItemA1 or ItemA2, but not both. Or, the user can select any number of items in the control, up to the number displayed.
- Control the quantity of the domain members themselves. The user or the computer can select one or more of ItemA1.
- Do both of the above in combination:
 - One or more of ItemA1 and ItemA2.
 - One or more of ItemA1 and just one of ItemA2.
 - One of ItemA1.
 - One or more of ItemA2.
- Specify default selections based on quantities—if the user selects three of ItemA1, then select ItemB1 (or 3 of ItemB1, if desired).
- Specify default quantities based on selections—if ItemA1 is selected, then select three of ItemB2 (and ItemB3 and ItemC1, if desired).
- Limit or specify the number of items a user can select within a selection point. For example, the user can select up to three of ItemA3, and no more than one of ItemA4.
- Set up minimum default quantities to ensure that enough of ItemA3 is ordered when it is selected.
- Ensure that enough of ItemA3 is ordered when the user selects ItemB2 in another selection point, which requires a specific quantity of ItemA3.
- Use an expression to calculate the required quantity of an item that is dependent on user input.
- Determine item quantities based on domain member values.

Advanced Configurator default quantity definitions allow you to build flexibility into quantity determination for domain members that takes into account both predetermined quantity requirements and their interaction with runtime conditions. The following illustrates the problem:



Quantity determination in response to runtime events

Advanced Configurator lets you set limits on the final quantities, whatever quantity choices the end-user makes:



Dynamic process of determining the final quantity of a selected domain member

You must be able to set limits on the quantity of each domain member, yet allow for quantities that result from runtime processes, such as constraints and user entries. In addition, you may need to take into account quantities defined for its selection point.

Defining quantities in models depends on these basic concepts:

- Quantity defaults for individual domain members control how many of the selected item is chosen—2 of ItemA, or 0 of ItemB.
- A quantity defined for an entire control—in the model represented by the selection point—determines how many of the control’s items can be selected. If the quantity is one, it is a single-select control; more than one, a multi-select control.

- You can set default quantities at design time whose default values do not change during run time, acting as absolute values for calculating a final quantity that takes into account quantities generated by run time events such as user picks and constraints.

Static Default Quantities

The quantity of the selection DomainMemberX can be affected by static default quantities—the default value you assign to it during modeling. An example is the quantities assigned automatically when a domain member is selected.

For example, in a network model, NodeY always requires at least two Routers, more if certain other components are selected in combinations determined by various constraints. By setting a static default of 2 on the NodeY domain member, and giving it a Quantity Policy of MIN, you can ensure that if it is selected, it will be ordered in a quantity of at least 2. Static default information for a selection point is applied whenever a dynamic default or Requirement constraint selects any domain member of that selection point, provided there are no user selections.

In addition, if the other nodes in the Node selection point—NodeX and NodeZ—have the same MIN(2) requirement, you can set the *Use Quantity Policy for all Domain Members* option as a “blanket” default.

See Also

[Chapter 3, “Understanding Modeling,” Interaction between Default Quantities and Min/Max Settings at Run Time, page 42](#)

Dynamic Default Quantities

The value of dynamic default quantities is determined at run time in response to expressions or dynamic default constraints. Expressions can be included in a quantity policy—*Overridable $f(x)$* , *Min of $f(x)$* , *Max of $f(x)$* , and *Sum $f(x)$* .

The Requirement constraint and Dynamic Default relationships select domain members in response to user or computer picks during run time. Quantity for the default-selected domain member is 1, unless you specify a new quantity and a new quantity policy. Quantities are defined on the table editor for the relationship.

A dynamic quantity definition would be needed in the case of a computer model. The model specifies by a dynamic default that when BoardA is selected, the Advanced Configurator should select FanA as well. Thus, a computer-select occurs for FanA, with a quantity of 1. (This computer-select occurs unless the selection causes a conflict with another selection that is either required or user-selected.) However, BoardB requires 2 of a different fan, FanB, so quantity must be added to the default definition of the BoardB/FanB requirement. In the relationship editor, the constraint’s RHS for BoardB would be *Min of 2* for FanB.

Only selection points with *Quantity = True* are settable.

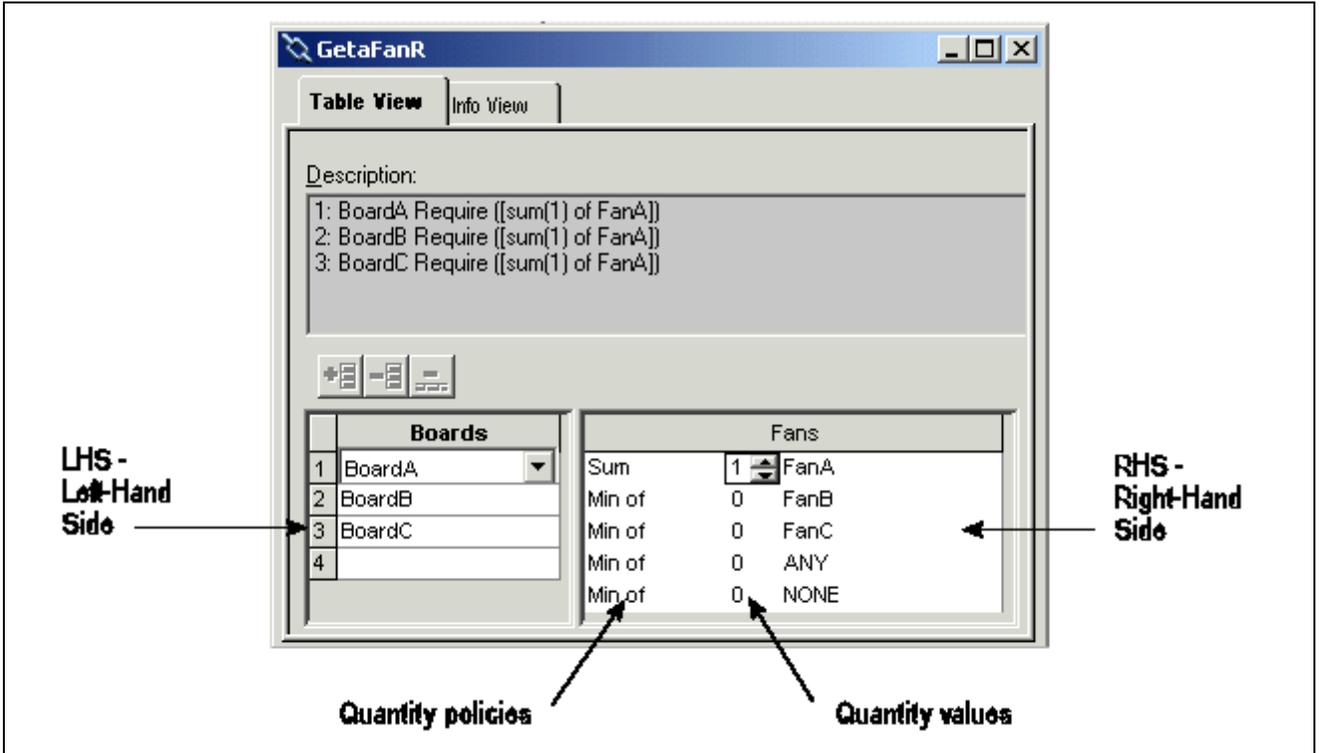
Multiple Selections on a Single Domain Member

When there is the potential for multiple default selections on the same domain member, you must set up contingent definitions at design time that correctly calculate the final quantity no matter what or how many of the domain members are selected in response to runtime criteria. An example is a network model that contains server boxes requiring chassis. The chassis type (domain member) and quantity depend on the number of slots the server requires, as one chassis provides two slots and the other chassis provides four. When the user selects the desired server boxes, the proper chassis type(s), in the required number, are default-selected. In such a case, the quantity of chassis’ can be the largest of the default quantities (minimum value), it can be summed, or it can be a combination of the two.

Order of Evaluation

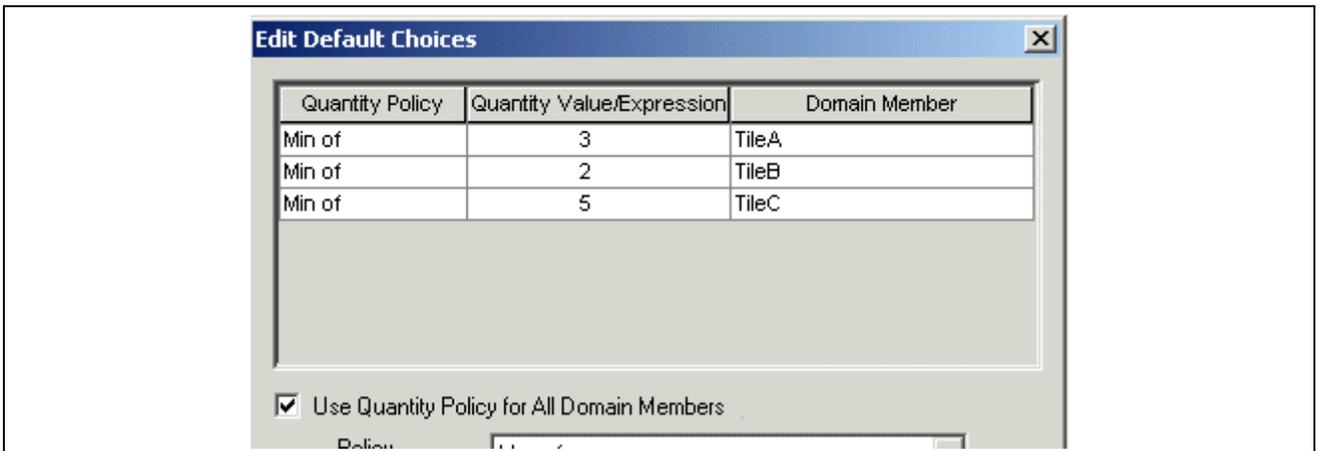
When a domain member has multiple default quantities, the Advanced Configurator evaluates the selections in a predetermined order.

Step 1—Default quantities for the domain members are evaluated first to arrive at a single value according to the policy: the least value (*Min of*); the greatest (*Max of*), and total (*Sum*). These are specified in the relationship editor:



Relationship editor showing quantity settings for BoardA

Step 2—The domain member’s static Min/Max setting, if any, is applied: adjust up to the minimum value (*Min of*), adjust down to the maximum value (*Max of*), total (*Sum*). If *Overridable* is the policy, this step is not performed. Policies are specified in the Defaults editor:



Default editor and static default settings

Step 3—If *Use Base Quantity Policy for All Domain Members* is specified, it is applied against the result of Step 2 in the same way as the domain member’s static policy.

The screenshot shows a dialog box with the following elements:

- A checked checkbox labeled "Use Quantity Policy for All Domain Members".
- A "Policy" dropdown menu currently showing "Max of".
- A "Quantity" spinner control currently showing the value "5".
- "OK" and "Cancel" buttons at the bottom right.

Default editor and dynamic default settings

Step 4—Step 3 yields the final calculated value. *If a user enters a value, it overrides this value.*

Example 1: Taking the largest quantity of the selected domain members

At Design Time

If you need to define default quantities for domain member A on a selection point so that the final quantity after runtime selections was at least n, the settings in the defaults editor are:

$$A_{\text{Default}} = \text{MIN}(2)$$

(The default minimum quantity for domain member A when a runtime selection is NOT made, or when a selection IS made but is less than n. Static default.)

$$A_{\text{User}} = \text{MIN}(x)$$

(The default minimum quantity for domain member A when the user selects A and enters a quantity for it (using a control with quantity). It differs from $A_{\text{Default}} = \text{MIN}(n)$ in that the default quantity is left to the user.)

$$A_{\text{DynamicDefault(B)}} = \text{MIN}(3)$$

$$A_{\text{DynamicDefault(C)}} = \text{MIN}(5)$$

(The default minimum quantity for A when domain member B or C is picked by an expression or constraint triggered during run time.)

Example 1: Taking the largest quantity of the selected domain members

Steps in calculating the final quantity:

- Selections are made.
- The runtime quantities for each A selection are compared and the largest is applied against the static default $A_{Default} = \text{MIN}(2)$.
- If it is equal to or greater than 2, the runtime value will be A's final quantity; if not, $A_{Default} = \text{MIN}(2)$ is applied, and A's final quantity is 2.

At Run Time:

If the user picks *none*, then $A = 2$.

If the user picks *B*, then $A = 3$.

If the user picks *C*, then $A = 5$.

If the user picks *B* and *C*, then $A = 5$.

If the user picks *A [4]* and *B*, then $A = 4$.

If the user picks *A [4]* and *C*, then $A = 5$.

Example 2: Taking the sum of the selected item(s)

At Design Time

Using the SUM policy, you can specify that the final quantity of a selected item is the sum of its default-selected quantities. As in the previous example, you can ensure that if no items are selected during run time, A will nonetheless be assigned a quantity value ($A_{Default}$).

$A_{Default} = \text{MIN}(2)$

(The default minimum quantity for domain member A when a runtime selection is NOT made, or when a selection IS made but is less than n. Static default.)

$A_{User} = \text{MIN}(x)$

(The default minimum quantity for domain member A when the user selects A and enters a quantity for it (using a control with quantity). It differs from $A_{Default} = \text{MIN}(n)$ in that the default quantity is left to the user.)

$A_{DynamicDefault}(B) = \text{MIN}(3)$

$A_{DynamicDefault}(C) = \text{MIN}(5)$

(The default minimum quantity for A when domain member B or C is picked by an expression or constraint triggered during run time.)

Example 2: Taking the sum of the selected item(s)

Steps in calculating the final quantity:

- Selections are made.
- The quantities of each A selection are totaled.
- The total is applied against the static default $A_{\text{Default}} = \text{MIN}(2)$.
- If it satisfies static default policy, the runtime sum is A's final quantity; if not, $A_{\text{Default}} = \text{MIN}(2)$ is applied.

At Run Time:

If the user picks *none*, then $A = 2$.

If the user picks *A*, then $A = 2$.

If the user picks *B*, then $A = 3$.

If the user picks *C*, then $A = 5$.

If the user picks *B* and *C*, then $A = 8$.

If the user picks *A* and *B*, then $A = 3$.

If the user picks *A* [4] and *B*, then $A = 4$.

If the user picks *A* [1] and *C*, then $A = 5$.

Example 3: Figuring quantities using attribute values and expressions

At Design Time

A's final quantity can be based on an attribute value or one derived from an expression. For example, using attributes to define Min/Max limits allows you to specify quantity limits on a per-domain member basis for external data. Expressions in defaults allow you to determine the quantity to default dynamically based on an external parameter, such as a value entered via an extern or an equation. Similarly, expressions in Min/Max limits enforce dynamic quantity limits.

The following example's definitions includes a dynamic default on domain member A that is the sum of all the values of the *A_Needed* attribute of the B domain members selected.

$A_{\text{Default}} = \text{MIN}(2)$

(The default minimum quantity for domain member A when a runtime selection is NOT made, or when a selection IS made but is less than n. Static default.)

$A_{\text{User}} = \text{MIN}(x)$

(The default minimum quantity for domain member A when the user selects A and enters a quantity for it (using a control with quantity). It differs from $A_{\text{Default}} = \text{MIN}(n)$ in that the default quantity is left to the user.)

$A_{\text{DynamicDefault}}(B) = \text{SUM}(B:A_{\text{Needed}})$

(The default quantity for A when domain member B is picked during run time. The quantity is the value of B's attribute *A_Needed*. The SUM policy indicates that the value is to be added to any other default value for A in calculating A's final quantity.)

$A_{\text{DynamicDefault}}(C) = \text{SUM}(5)$

(As above, with the value predetermined at design time to be 5.)

$A_{\text{DynamicDefault}}(D) = \text{MIN}(\text{exp_name})$

(The default quantity for A when domain member D is picked during run time. The quantity is the value of an expression defined in the Expression editor. The MIN policy indicates that the final value must be equal to or larger than the value given by the expression.)

$(\text{exp_name}) = 3$

B1, $A_{\text{Needed}} = 1$

(An instance of B where the attribute *A_Needed* has a specific value.)

B2, $A_{\text{Needed}} = 2$

B3, $A_{\text{Needed}} = 3$

B4, $A_{\text{Needed}} = 4$

Example 3: Figuring quantities using attribute values and expressions
<p><i>Steps in calculating the final quantity:</i></p> <ul style="list-style-type: none"> • Selections are made. • Add the values for ItemA's SUM policy selections. • Take any values with MAX policies and compare them against the specified MIN values, including the static default $A_{Default} = MIN(2)$ to check that the runtime value satisfies the minimum quantity requirements. • If the MIN requirements are met, apply the calculated runtime quantity. If MIN requirements are not met, apply the quantities specified for the MIN defaults.
<p><i>At Run Time:</i></p> <p>If the user picks <i>B1</i>, then $A = 1$.</p> <p>If the user picks <i>B2</i>, then $A = 2$.</p> <p>If the user picks <i>B1</i> and <i>C</i>, then $A = 6$.</p> <p>If the user picks <i>B4</i> and <i>C</i>, then $A = 9$.</p> <p>If the user picks <i>A[4]</i> and <i>B3</i>, then $A = 4$.</p> <p>If the user picks <i>A[3]</i> and <i>B4</i>, then $A = 4$.</p> <p>If the user picks <i>A</i> and <i>D</i>, then $A = 3$.</p> <p>If the user picks <i>B1</i> and <i>D</i>, then $A = 3$.</p> <p>If the user picks <i>B2</i> and <i>C</i> and <i>D</i>, then $A = 7$.</p>

Understanding Minimum and Maximum Selections and Limits

Advanced Configurator allows you to set constraints based on the minimum and maximum quantity entered for domain members and on the minimum and maximum number of selections made within a selection point.

Min/Max limits can be specified on these Visual Modeler objects:

- Selection points
- Domain members

Selection Point Min/Max

Min/Max limits on a selection point determines whether its control is single-select or multi-select, and if multi-select, how many of the domain members can be selected.

You can use any of the following to input min/max limits:

- An absolute number (for a static, known check).
- A database reference.
- A reference to an expression defined elsewhere in the model.

Note. With dynamic inputs such as the result from a database query or an expression, it is possible for a single-select control to be re-specified a multi-select control during a configuration session when the SP Max changes from 1 to >1. When the Max limit is specified in this way, the control will behave like a multi-select that is limited to a single selection, versus a true single-select. Limitations that apply to multi-select selection points will apply even when the maximum number of selections is one. For instance, multi-selects do not allow violation explanation substitutions, and there are limitations on how they can be used in constraints.

The screenshot shows three panels related to configuring a selection point control:

- Property Table Editor:** A table with columns Name, Type, and Value.

Name	Type	Value
Name	String	Fans
Filename	String	C:\Program File
Type	String	Fans
Quantity	Bool	False
Use Min/	Bool	True
SP Min/M	Editor	Edit...
Defaults	Editor	Edit...
- Min/Max Dialog for Selection Points:** A dialog box titled "Selection Point Min/Max For Fans". It has two sections:
 - Selection Point Minimum Selections:** Radio buttons for Number (selected, value 0), Expression, and SQL Query. Fields for Database and Explanation.
 - Selection Point Maximum Selections:** Radio buttons for Number (selected, value Unbounded), Expression, and SQL Query. Fields for Database and Explanation.
- Resulting Control:** A control titled "Fans" with three checked items: Fan90_120, Fan90_240, and FanM90_240.

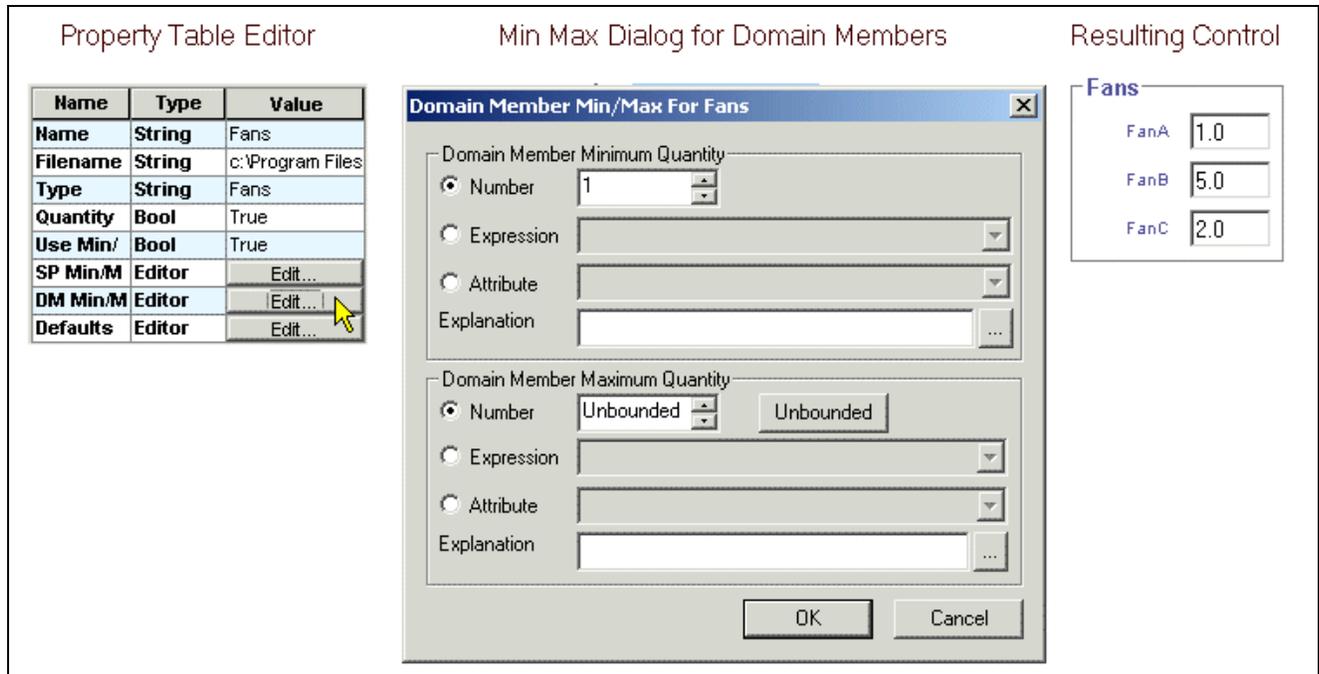
Selection point Min/Max

Domain Member Min/Max

Min and Max limits on domain members determine how many of each domain member can be ordered if selected.

Values for domain member Min and Max limits are input from:

- An absolute number entered by the modeler.
- A reference to an expression.
- A domain member attribute whose type is Int or Float.



Domain member Min/Max

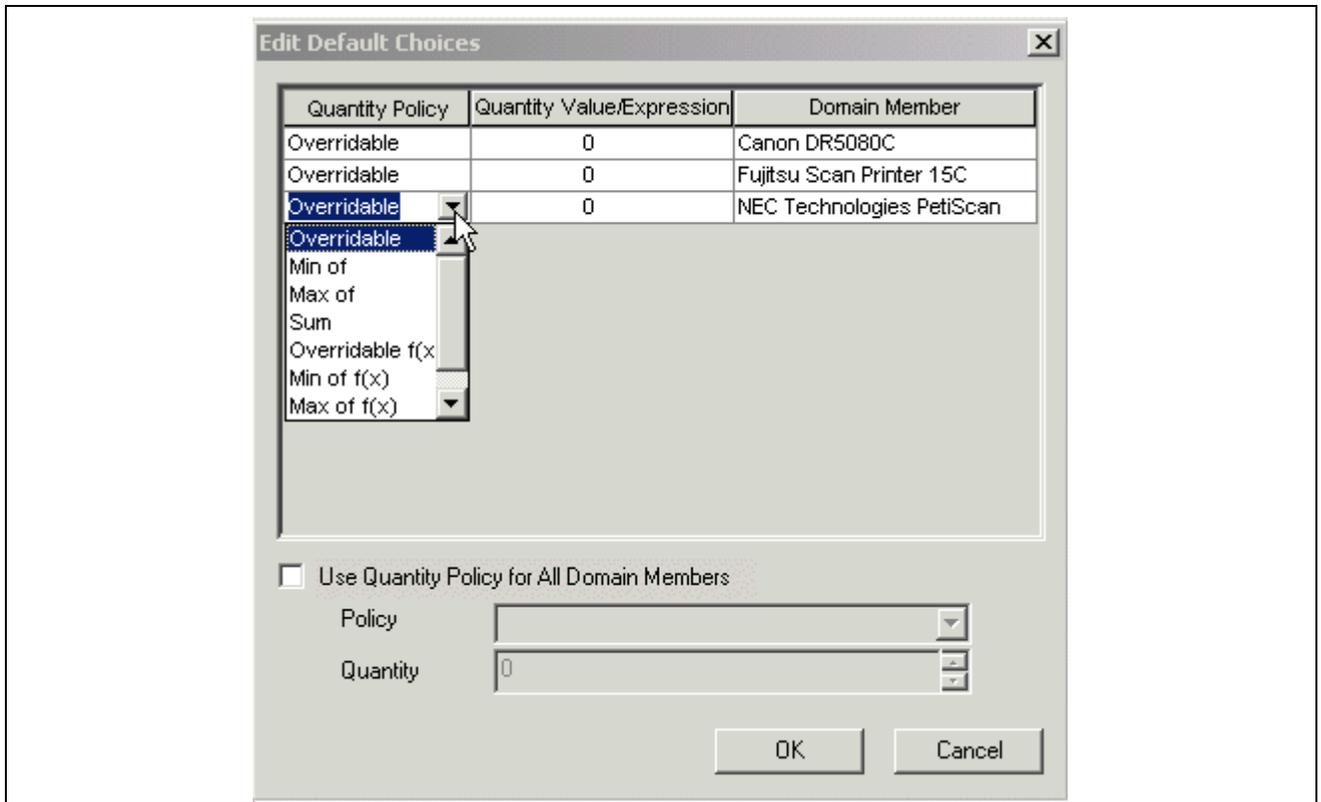
Note. The Min/Max function applies to the number of selections on a selection point that can be specified at the selection point level or the quantity that can be entered for a selected domain member at the domain member level, or both. It is a separate function from the Min of and Max of quantity policies of the Default Quantity functionality described in the previous section.

Interaction between Default Quantities and Min/Max Settings at Run Time

Min/Max settings add another aspect to the runtime process of determining final quantities on selected domain members.

When a selection point has both default quantity properties and Min/Max properties defined on it, specific runtime behaviors apply.

The Quantity policy—the manner in which to apply the default value—for a static default can be either Overridable, MIN, MAX, or SUM. If MIN, MAX or SUM is specified, then the static default is used in conjunction with any dynamic defaults and requirement constraints that select the same domain member. By default, the Quantity Policy for a static default is Overridable (meaning, “do not apply the value if run time selections assign a quantity”).



The quantity policy applied to the domain members of a selection point

Static default information for a selection point is applied whenever a dynamic default or a requirement constraint selects any domain member of that selection point and there are no user-selects. A static default on a specific domain member applies only to that domain member.

Default quantities, whether static or dynamic, are not applied if they would cause any constraint to be in violation.

Ultimately, a user-specified quantity overrides any quantities specified by static defaults, or assigned by dynamic defaults during run time. Advanced Configurator allows the user to enter a quantity even if it violates a constraint, a quantity default, or min/max settings. However, the appropriate violation explanation will be displayed.

Note. When you use the Model Tester, you may observe that when you attempt to undo or back out of a configuration pick sequence, all picks on the control disappear rather than just the expected last pick. This occurs because the Configuration Engine is stateless; it receives all picks at one time, at each submission, and doesn't know which was the last pick. By turning off Auto-Submit when you test the selection point on the Model Tester, you can observe true runtime behavior.

Minimum Violation Explanation and Incomplete Configuration Explanation

Advanced Configurator distinguishes between two very similar violation circumstances involving quantities on a selection point.

The project setting “Incomplete Configuration Explanation” is similar to the Min/Max violation explanation in that it alerts the user of a non-valid configuration condition.

These two explanations differ in the circumstances in which they are presented.

If a completeness check operation is requested, the Incomplete Configuration Explanation is shown when no selections have been made on a required selection point.

If selections have been made on the selection point, but the number of selections doesn't satisfy the minimum value specified, the selection point Minimum Selection Explanation will be shown rather than the Incomplete Configuration Explanation.

See Also

[Chapter 4, "Setting Up the Modeling Environment," Specifying Model Project Settings, page 60](#)

Creating Parameterized Explanations

Advanced Configurator allows you to offer specific information to users when their picks violate a constraint based on a numeric relationship. Using the parameters representing objects such as maximum quantity allowed, you can write a violation explanation that describes the violation more specifically than does a generic text message.

You can include parameters in the Explanation fields of all constraint editors, and in the SP and DM Min/Max editor dialogs.

You can also represent external objects.

If the control is single-select, you can parameterize an explanation specific to the domain members in violation. Parameter substitution can not resolve the domain members in multi-select controls, so use on multi-select controls is limited to the selection point level Min and Max Value, selection point Name, or an expression value.

General syntax is:

```
$(replacement_specification)
```

replacement_specification is either

```
n:attribute_identifier or expr:expression_identifier.
```

n is the positional identifier for the class within the relationship ($0 - n$ left to right in the relationship).

Note. Use $n = 0$ for SP Min/Max and DP Min/Max explanations.

attribute_identifier is a class attribute name or one of the reserved names from the list in the table below. If the objects are external, you must refer to the database column name instead of an attribute name.

expression_identifier is an expression name.

The following table describes the reserved parameters and indicates whether each is available for single- and multi-select controls.

Note. Please observe letter case as shown in the syntax column for each variable (expression and attribute names are lower case; all others are upper case).

Parameter name	Syntax	Single-select control	Multi-select control
Selection point name	\$(n:\$DPNAME)	yes	yes

Parameter name	Syntax	Single-select control	Multi-select control
Selection point Min Value	\$(n:\$MINCHOICES)	yes	yes
Selection point Max Value	\$(n:\$MAXCHOICES)	yes	yes
Domain member name	\$(n:\$NAME)	yes	no
Current number of selections made in the Selection Point	\$(n:\$CHOICES)	yes	no
Selected domain member Min Value	\$(n:\$MINQTY)	yes	no
Selected domain member Max Value	\$(n:\$MAXQTY)	yes	no
Selected domain member quantity	\$(n:\$QTY)	yes	no
Domain member attribute value	\$(n:attributename)	yes	no
Expression value	\$(expr:exprname)	yes	yes

For example, in a requirement constraint, messages are:

Explanation syntax The selected base requires the \$(1:Description) power cord and \$(2:Description) chassis.

Run-time display The selected base requires the Z40-15 power cord and the 8R_KU chassis.

In another example, in a DP Min, messages are:

Explanation syntax The value of \$(0:\$DPNAME) must be between \$(expr:refract_min) and \$(expr:refract_max).

Run-time display The value of LensRefraction must be between 1.0 and 4.0.

Optimizing Performance and Minimizing Model Maintenance

To improve performance and minimize effort expended on maintaining a model, consider these tips:

- Plan your model hierarchy so that you can use attributes efficiently.
- Stabilize attribute names, class names, and selection point names before creating constraints or expressions.
- If a model uses numerous multi-select controls, it is possible for constraints against the same control to conflict with other constraints to the point that all items on a control are eliminated. The following practices can reduce the occurrence of this problem:

- Articulate the class hierarchy to a greater degree. If a multi-select control contains many domain members that are known to conflict, create subclasses and group the domain members in compatible sets. Alternatively, separate single-select and multi-select items.
- Write more constraints. Instead of writing a single large constraint with many columns and rows, write more specific classes and constraints.
- Do not use a directional compatibility constraint if a multi-select is on the right-hand side. A directional compatibility with a multi-select on the RHS is ignored because every combination is considered valid. If a multi-select is required on the RHS, write the constraint as a directional non-compatibility constraint or a Requirement constraint.
- If possible, use a Requirement constraint instead of a Compatibility constraint. Because a Requirement constraint does not eliminate, there is less chance of conflicts between constraints.
- For all directional constraints (compatible, non-compatible, requirement, and dynamic default), if there are multiple arguments on the RHS, there is no explicit relationship between them. So, instead of writing multiple constraints against a single class, it is more convenient to write a single constraint with multiple arguments on the RHS.

Using the Sample Models

Visual Modeler is shipped with these example models:

- A component model called Sample, which you can load by choosing the Sample button in the Visual Modeler launch screen.
- A telecommunications compound model called TelcoSampleCompoundModel, containing three component models TelcoComp, TelcoCompCircuit, and TelcoCompHub.

The Modeling Process

Defining a robust model of a product or service requires some or all of these steps:

1. Use Visual Modeler to:
 - Build a class structure that represents data relationships in a configuration problem.
 - Define class attributes, create domain members, and provide attribute values in the model, or bring in domain members and attribute values from an external source, such as a database or the user.
 - Write relationships between classes and class attributes on a component model.
 - Specify quantities for default selections and define quantity behaviors that calculate a quantity at run time.
 - If supported version control software is available and connected in the Visual Modeler, you can interact with the version control software at any time during model development.
 - If you are building a compound model, create configurable components and associate each with a component model, then create the relationships between the component models.
2. Connect to the Configurator engine to compile the model and launch the Model Tester.
3. In the Model Tester, verify that relationships work properly on runtime controls.

4. Using model information from the Visual Modeler source, use PeopleSoft Extensions for DreamWeaver to develop JSP pages or use those provided with PeopleSoft Order Capture.

See Also

Part 5, “PeopleSoft CRM Order Capture Integration,” page 271

Part 6, “Building a Custom User Interface,” page 295

Model Tester

Included with the Visual Modeler install is a web-based test tool that renders the selection points in a pre-formatted form when you compile and run the model. Using it, you can test most facets of model behavior without building your own test UI. The Model Tester lets you verify the validity and behavior of relationships, data input and handling, error recognition, and control behavior.

See Chapter 4, “Setting Up the Modeling Environment,” Using the Model Tester, page 66.

Interfacing with Third-Party Tools

This section lists Advanced Configurator interfaces with industry-standard applications.

Microsoft SQL Server, Oracle Databases, and IBM DB2

Visual Modeler can query tables in Microsoft® SQL Server™, Oracle™, or IBM® DB2® databases. In addition to obtaining domain members from a database, you can write model constraint information to a database table, or read constraint information from a database.

Macromedia DreamWeaver

Visual Modeler creates an XML file that contains information about the model that can be used to lay out controls at run time. Advanced Configurator provides extensions to Macromedia® DreamWeaver® so that you can use this information to create JSP pages.

See Chapter 28, “Using the Page Editor Extensions for Dreamweaver,” page 359.

Source Control Interfaces

The Visual Modeler uses a standard Microsoft interface to access compatible configuration management software. Microsoft® Visual SourceSafe™, Rational ClearCase®, and Merant™ PVCS are verified as being compatible.

See Chapter 4, “Setting Up the Modeling Environment,” Source Control Software, page 50.

CHAPTER 4

Setting Up the Modeling Environment

This chapter discusses:

- Connecting to third-party software.
- Connecting to a database from Visual Modeler.
- Getting started with Visual Modeler.
- Understanding project files.
- Importing and exporting models.
- Compiling a model.
- Using the Model Tester.
- Interfacing with third-party tools.

Common Elements in this Chapter



Open the Overview Window.



Save and compile the model.



Compile and run the model.



Open the Viewer dialog, which contains the Find tab, where objects are listed.



Standard Windows “show and hide” indicator buttons. Appear in the model structure view to control display of hierarchy items. Also indicate that the domain members of the node are internal.



In the model structure view, this icon appears next to external domain members. Internal domain members are indicated by the plus and minus symbols.



Add button for the component properties table. Adds rows—properties—to a class or subclass.



Add button for the component properties table. Adds a row—properties—for defining attributes on selection points.

Connecting to Third-Party Software

This section discusses setup procedures for:

- Source control software.
- Database interface configuration.
- Connecting to a database from Visual Modeler.

Source Control Software

The Visual Modeler supports the latest version of the Microsoft Source Control (MSC) interface. An installed source control application that complies with the correct MSC version is shown as an option in the Visual Modeler.

1. Create or obtain a user account and login for your source control tool.
2. In Visual Modeler, go to Tools, Options, then click the Source Control tab.
3. Choose a provider from the “Source Control Provider to Use” drop down. This drop down displays MSC-compatible applications installed on your system.

Optionally, select the source control options best suited to your project:

Option	Explanation
<i>Get files when opening the workspace.</i>	Automatically get the latest versions of all .cms files in this project whenever a workspace is opened.
<i>Check in files when closing the workspace</i>	If this option is checked, you must remember to save before closing the workspace. If files are saved before the workspace closes, the all files will be checked in. If unsaved files are open when the workspace is closed, the files will be checked in, but any files previously not saved will be checked out again.
<i>Prompt to add files when inserted</i>	When a .cms file is inserted into the project, prompt to add the file to source control.
<i>Perform background status updates</i>	Not all source control tools support this option. If background status updating is supported, the IDE can change the appearance of a file in the File View when a file under source control is altered by an external checkin.
<i>Use dialog for checkout</i>	Specify that a comment dialog automatically appears when a file is checked out.
<i>Add new projects to source control</i>	Automatically add new projects (.csp file) to source control.
<i>Automatically add new files to source control</i>	Automatically add new .cms files to source control.

4. Enter the source control login name in the Login field, then click OK. You will be notified that the provider change will not take effect until you restart the Visual Modeler. The changes will be available when the Visual Modeler is restarted.

Once the source control tool is selected, it is possible to set Advanced options. Select Tools, Options then click the Source Control tab. Click the Advanced button. An Options window appears. Consult your source control documentation for information on the options displayed.

Database Interface Configuration

Advanced Configurator supports Microsoft SQL Server, Oracle, IBM DB2 database connections.

Note. If your database is Oracle, you must install an Oracle client on the machine where Visual Modeler is installed. Your system will then have the Oracle ODBC Driver, which ensures compatibility with Oracle.

Before proceeding, you need certain system information:

- The database name, username/login, and password. Consult your database administrator for login and password information. At a minimum, you must have read and write permission for the database.
- The name (machine ID) of the machine that the database server resides on, and the port it uses.
- The name of the machine that WebLogic is installed on, and the port it uses. If the defaults were accepted during Advanced Configurator installation, the default port number is 7777.

The Visual Modeler relies on the ODBC Data Source Administrator for connection information. Setting up an ODBC data source allows you to view external domain members in the Visual Modeler.

See [Chapter 4, “Setting Up the Modeling Environment,” Connecting to a Database from Visual Modeler, page 54.](#)

To compile a model, Advanced Configurator uses the Java Naming and Directory Interface (JNDI) Database connection. This connection requires customizing two properties files.

See [Chapter 4, “Setting Up the Modeling Environment,” Configuring JNDI DBName.properties, page 53.](#)

Configuring an ODBC Data Source for Microsoft SQL Server

Use the Windows data source wizard to configure an ODBC data source.

To connect Advanced Configurator to Microsoft SQL Server:

1. Open My Computer, Control Panel, Administrative Tools, and Data Sources.
2. Click the Add button.
3. Select a driver from the list, then click Finish. The data source connection dialog appears.
4. Enter the indicated information in the fields. Choose a name carefully. The name is case-sensitive, and it will be used to identify the data source both in the properties files and in the Visual Modeler database interface.
5. Click Finish. The authentication dialog appears.
6. Select an authentication option. If a password is not required, click “With Windows NT authentication of the login ID”, then click Next.

If the database has a login and password, click “With SQL Server authentication using a login ID and password entered by the user,” then click “Connect to SQL Server to obtain default settings for the additional configuration options.” Enter the database login and password, then click Next. The remaining screens address DSN Configuration options that are not critical for the connection. Click Next, then on the last screen click Finish.

7. Configure JNDIDBName.properties.

See [Chapter 4, “Setting Up the Modeling Environment,” Configuring JNDIDBName.properties, page 53.](#)

Configuring an ODBC Data Source for Oracle

Use the Windows data source wizard to configure an ODBC data source.

To connect Advanced Configurator to an Oracle data source:

1. Open My Computer, Control Panel, Administrative Tools, and then Data Sources.
2. Click the Add button.
3. Select a driver from the list, then click Finish. The data source connection dialog appears.
4. Enter the indicated information in the fields. Choose a name carefully. The name is case-sensitive, and it will be used to identify the data source both in the properties files and in the Visual Modeler database interface. In most cases it is appropriate to accept the default settings for Database Options, Application Options, and Translation Options. Consult your database administrator. Click OK.
5. Configure JNDIDBName.properties.

See [Chapter 4, “Setting Up the Modeling Environment,” Configuring JNDIDBName.properties, page 53.](#)

Configuring an ODBC Data Source for IBM DB2

To connect Advanced Configurator to an IBM DB2 data source:

1. Launch the IBM DB2 setup tool (Programs, IBM DB2, Set-up Tools, Configuration Assistant).
2. Select Selected, Add Database Using Wizard. The wizard appears.
3. In window 1, Source, select Manually configure a connection to a database and click Next.
4. In window 2, Protocol, select your protocol.
5. In window 3, assign the appropriate values to the fields.
6. In window 4, Database, enter the database name and alias.
7. In window 5, Datasource, select Register this database for ODBC and As system data source. Enter a value for Data source name.
8. In window 6, Node Options, specify the operating system and the remote instance name.
9. In window 7, System Options, specify System name, Host name, and Operating system.
10. In window 8, Security Options, specify Use authentication in server’s DBM Configuration. Click Finish. The wizard closes and the Configuration Assistant reappears with the connection listed in the pane.
11. Select Selected, Test Connection to verify the connection.

Configuring JNDIDBName.properties

By default JNDIDBName.properties is located on the Advanced Configurator Server:

```
C:\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\Web-inf\config
```

Note. You may not need to hand-edit this file. If you are using data from the database specified during installation the file will be updated based on the information entered with a (datasource) name of “PSCFG.”

Only the portion of the file relevant to your database needs to be changed. All entries except the machine name are case-sensitive and must match your environment exactly. Substitute your information as follows:

1. Replace *sqldb*, *oracledb* or *DB2DB* with your data source name. This name is defined when the ODBC driver is configured. The “name”(the portion before the first ‘.’) should match the name of the data source specified for the classes in the model.

See [Chapter 4, “Setting Up the Modeling Environment,” Database Interface Configuration, page 51.](#)

2. Replace username and password with the user name or login defined for your database account. The square brackets [] used to delimit the sample must be deleted.

If you specified “Windows NT authentication using the network login ID,” the fields may be left blank.

3. Save a copy of the customized.properties file in a location outside the Advanced Configurator tree. The properties file will be overwritten if an upgrade is installed.
4. Bounce the server so that JNDIDBName changes can take affect.

The JNDIDBName properties file is shown below.

```
# example for setting up a SQL server database with NewAtlanta driver
# SQL server default PortNumber is 1433
sqldb.url=jdbc:JTurbo://[ServerName]:[PortNumber]/[DatabaseName]/sql70=
true=>
/charset=Cp1252
sqldb.driver=com.newatlanta.jturbo.driver.Driver
sqldb.username=[username]
sqldb.password=[password]

# example for setting up an Oracle database with Oracle driver
# Oracle default PortNumber is 1521
oracledb.url=jdbc:oracle:thin:@[HostName]:[PortNumber]:[OracleSID]
oracledb.driver=oracle.jdbc.driver.OracleDriver
oracledb.username=[username]
oracledb.password=[password]

# example for setting up a DB2 database with the Weblogic driver
db2db.url=jdbc:db2://[HostName]:[PortNumber];databaseName=[DatabaseName]
db2db.driver=weblogic.jdbc.db2.DB2Driver
db2db.username=[username]
db2db.password=[password]

PSCFG.url=jdbc:JTurbo://PSMITH081793:1433/Configurator/sql70=true/charset=Cp1252
PSCFG.driver=com.newatlanta.jturbo.driver.Driver
PSCFG.username=ConfiguratorUser
PSCFG.password=
```

Connecting to a Database from Visual Modeler

Once the ODBC data source and the properties files have been configured, your environment is ready to support a database connection within Visual Modeler.

Database connection involves two steps:

- Specify a database connection.
- Specify a default database.

See Also

[Chapter 4, “Setting Up the Modeling Environment,” Database Interface Configuration, page 51](#)

[Chapter 4, “Setting Up the Modeling Environment,” Configuring JNDIDBName.properties, page 53](#)

Specify a Database Connection

To specify a database connection:

1. In the Visual Modeler, select Project, Database References. A dialog for specifying the default database appears.
2. Click Add. A dialog for specifying or modifying a database connection appears.
3. Type a name in the Alias field. This is a name of your choice for referring to the database from within this model; it will be displayed in the drop down on the SQL query dialogs.
4. Type in the Data Source Name exactly as specified in the data source configuration .
5. Type in the Login ID and password defined for the database. These fields can be left blank if the database uses the Windows NT login. Press OK to view a dialog for setting the default database. The dialog for selecting the default database reappears, now displaying the database alias you specified.
6. To add another database, repeat steps 2 through 5.

Specify a Default Database

To specify a default database:

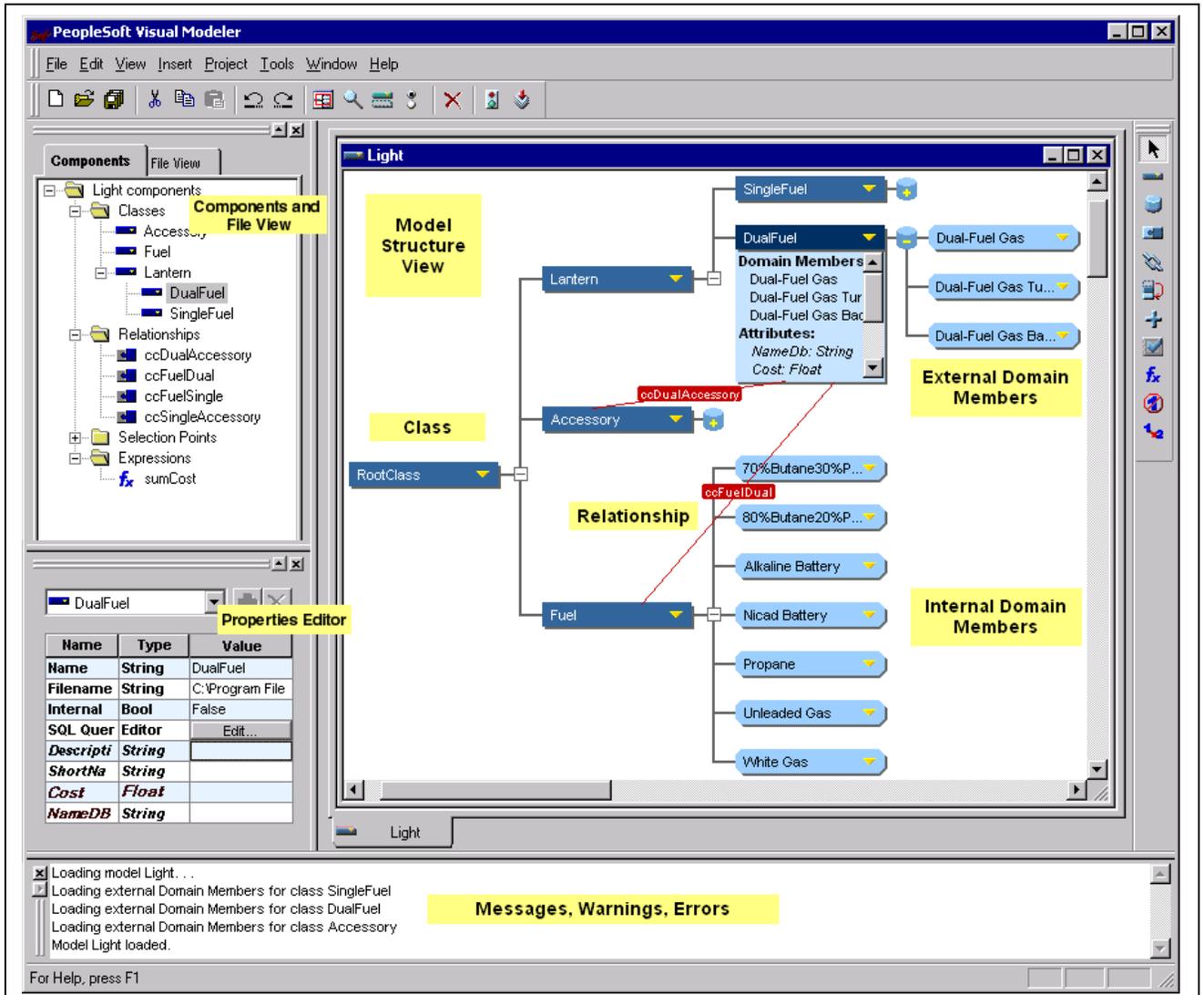
1. In Visual Modeler, select Project, Database References.
2. Select a the desired database in the database listing. The default database is the same for all database connections in the model.
3. Make sure the database name appears in the Default Database field at the bottom. Click OK.
4. Map a class to a database column.

See [Chapter 5, “Creating Objects for the Model,” Selecting a Primary Table, page 75](#).

The ability to specify a default is a convenience feature. If, at a later time, you need to query a different database, simply redefine the datasource name to refer to the new database.

Getting Started with Visual Modeler

To start Visual Modeler, select Start, Programs, PeopleSoft Applications, Visual Modeler 9.0, Visual Modeler 9.0. By default, Visual Modeler displays model information in three major windows. The largest area, the model structure view, is the modeling work area.

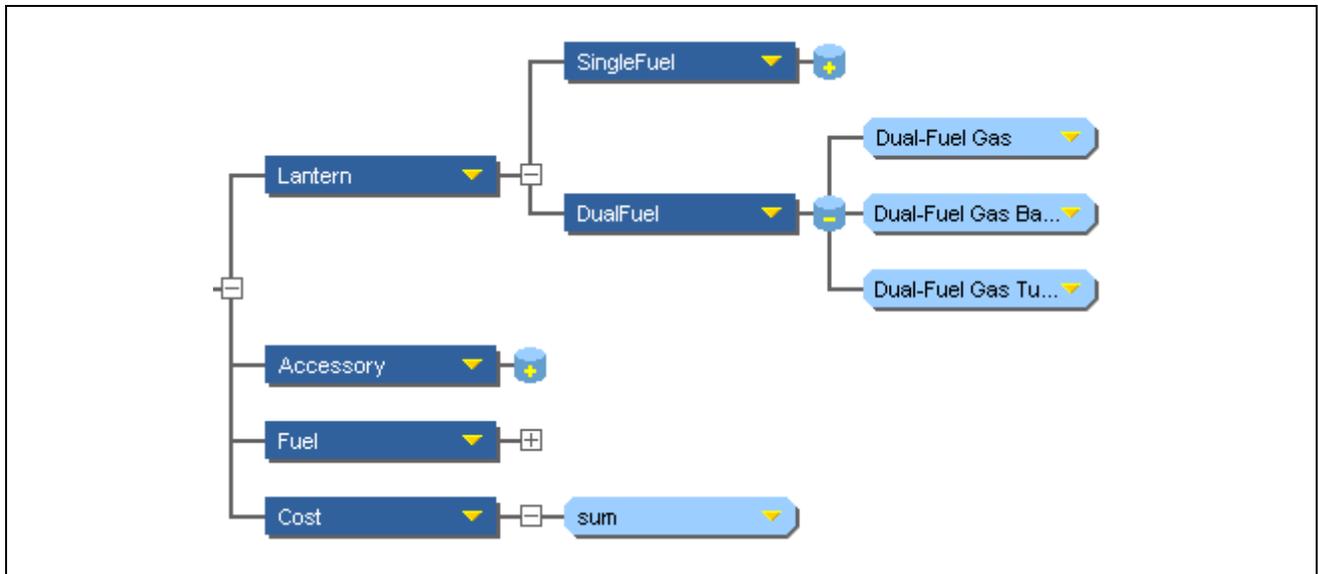


Visual Modeler window elements and model components

Model Structure View

Visual Modeler provides several tools for viewing and navigation in the Model Structure View.

The model structure view offers a standard windows “show and hide” paradigm for nodes in the class hierarchy. Click the “minus” icon to hide a hierarchy and click the “plus” icon to show all items in the hierarchy. If the node has external domain members, the database icon appears rather than the simple plus/minus icons.



Hierarchy with shown and hidden nodes

From the View menu you can Show or Hide the following:

- Selection Points
Equivalent to clicking the selection point icon.
- All Domain Members
Show/Hide all domain members in the model.
- Selected Class' Domain Members
Show/Hide the domain members for a class that is selected in the model structure view or in the Components view.
- Children
Show/Hide the class hierarchy for a class that is selected in the model structure view or in the Components view.

The View menu also offers similar options for expanding or collapsing items throughout the model.

Note. Expanding is equivalent to clicking the yellow triangle on an object.

Components and Files View

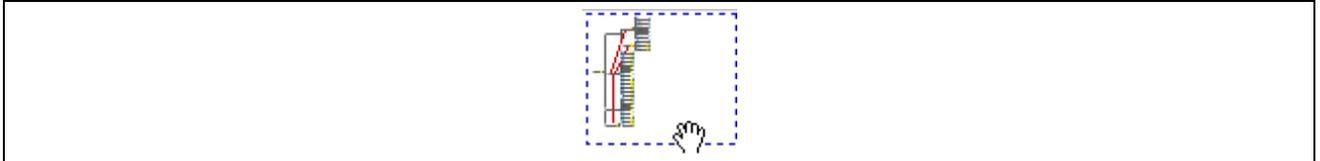
The Components tab is a navigation aid that displays the model's objects in hierarchical structure. The tab shows classes, subclasses, domain members, selection points, relationships, and expressions. Double-click on a class, domain member, or selection point to display it in the Model Structure view. Double-click on a relationship or expression and its editor appears. Properties for the selected object appear in the Properties Editor.

Properties Editor

All objects' properties appear in the property table when they are selected. Use this table-based editor to view, change, add, and delete an object's properties.

Overview Window

If the model is large, use the Modeling editor scroll bars to adjust the view. To “zoom” the model, click the Viewer icon to display the Viewer window Overview tab. A miniature view of the model is shown; the viewable area is outlined with a blue dashed line.



The overview window

To zoom the view, move the slide on the right.

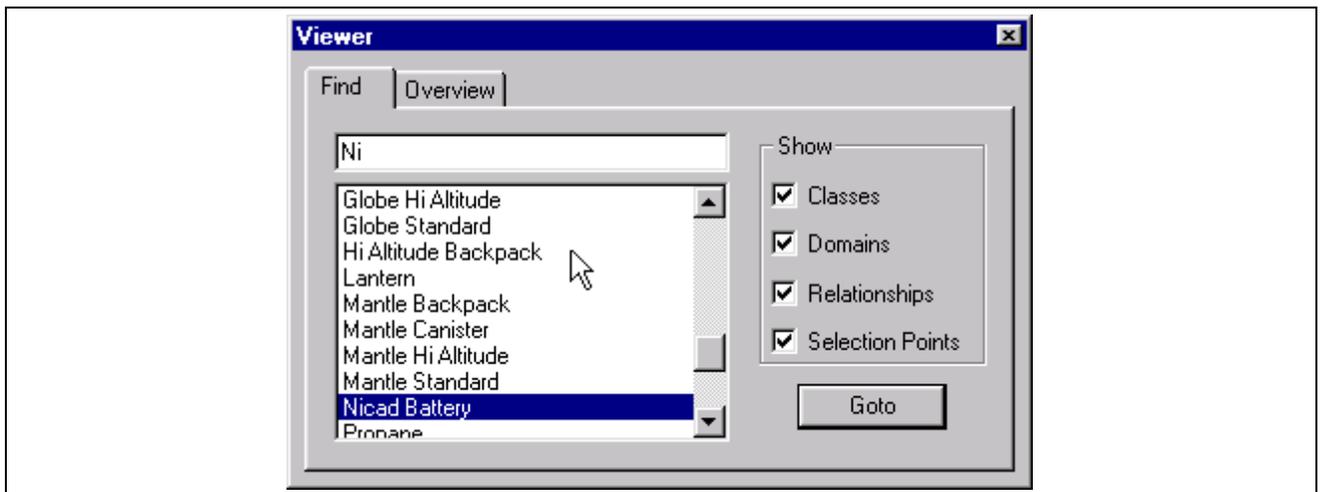
To refocus the model structure view display, click within the view outline; a hand cursor appears. Drag to reposition the view outline over the model.

Find Window

To find an object by name, click the Find icon, or select the X icon, then click the Find tab. Select or deselect the Show options to vary the objects displayed.

To find an object:

- Scroll the listing until you find its name.
- Start typing a name in the field above the listing. This field supports type-ahead, so the list will automatically jump to a partial match as you type.
- Select an item in the listing and click Go To, or type *Return* to bring the entry within the current window focus. Alternatively, double-click on an item.



Find window

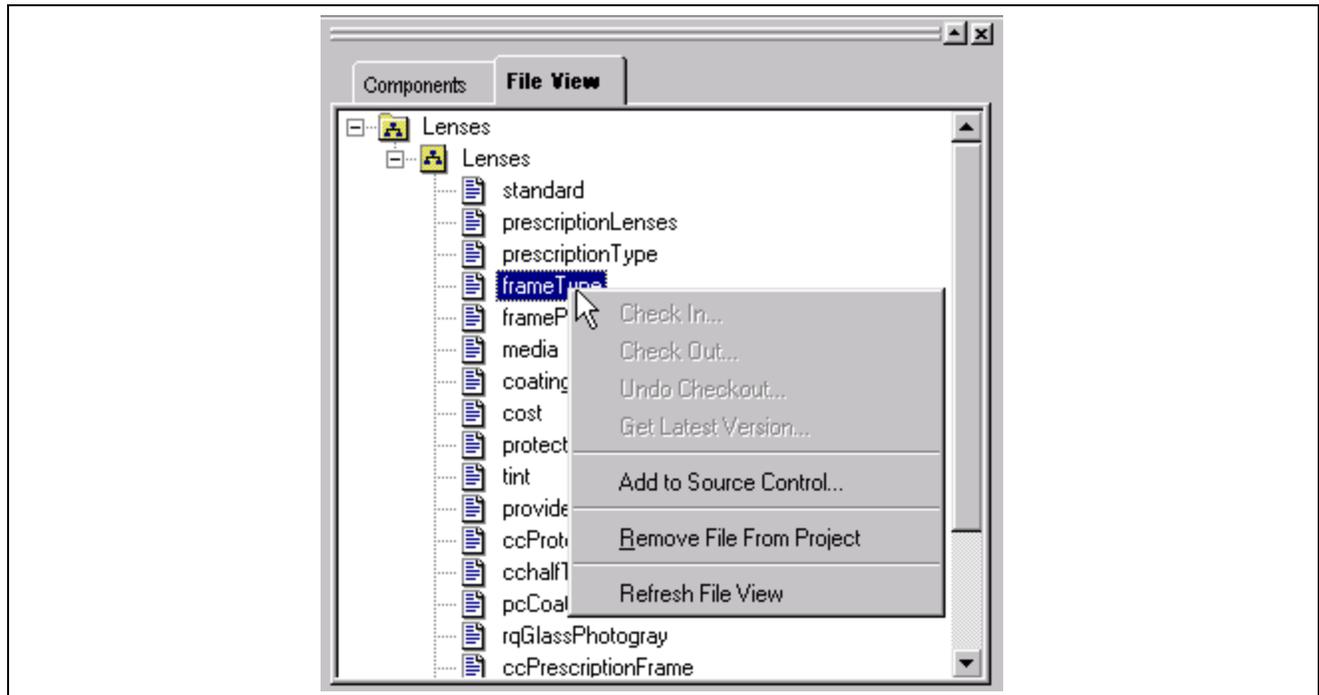
Understanding Project Files

A Visual Modeler project has three file types: a project file (.csp), a workspace (.csw), and multiple source files (.cms).

Configurator Source Project (.csp)

When a new project is created, a project file is written to the specified workspace. Typically the workspace and the project file share the same root name, but a single workspace can contain multiple projects if so desired. A Project file lists the .cms files required to describe the model, including standard.cms and any objects stored in separate .cms files.

The .csp file is maintained through Visual Modeler interaction. The File View displays the contents of the .csp file.



The File View

As you create objects, filenames are dynamically added to the .csp. Note that deleting an object in the model structure view does not remove its name from the .csp. To remove a file, right-click on a file to raise the context-sensitive menu. Removing a file from the project does not delete it from the workspace. Note that you can also perform source control operations from this menu.

See [Chapter 3, “Understanding Modeling,” Source Control Interfaces, page 47.](#)

Configurator Source Workspace (.csw)

The Visual Modeler displays one workspace at a time. Each workspace contains one or more projects. When a .csw file is opened, all the projects within it are opened. To view a different workspace, select File, Open Workspace to browse for a workspace, or, select File, Recent Workspace and select a previous workspace. If the current workspace has unsaved changes, you will be given the opportunity to save.

Configurator Model Source (.cms)

Advanced Configurator model source (.cms) files contain an XML representation of one or more model objects, for example, a class or relationship that is created in and saved from Visual Modeler. There can be many .cms files in a project.

By default, Visual Modeler creates a separate file for each class, relationship, and expression. At creation time, Visual Modeler proposes a .cms file name based on the object name. You can specify an alternate new name or an existing file name. If an existing file is specified, the new object is appended to it. A file name can only be specified at creation time; it cannot be changed after the fact. This implies that although the class name is changed in the Visual Modeler, the supporting file name is not affected.

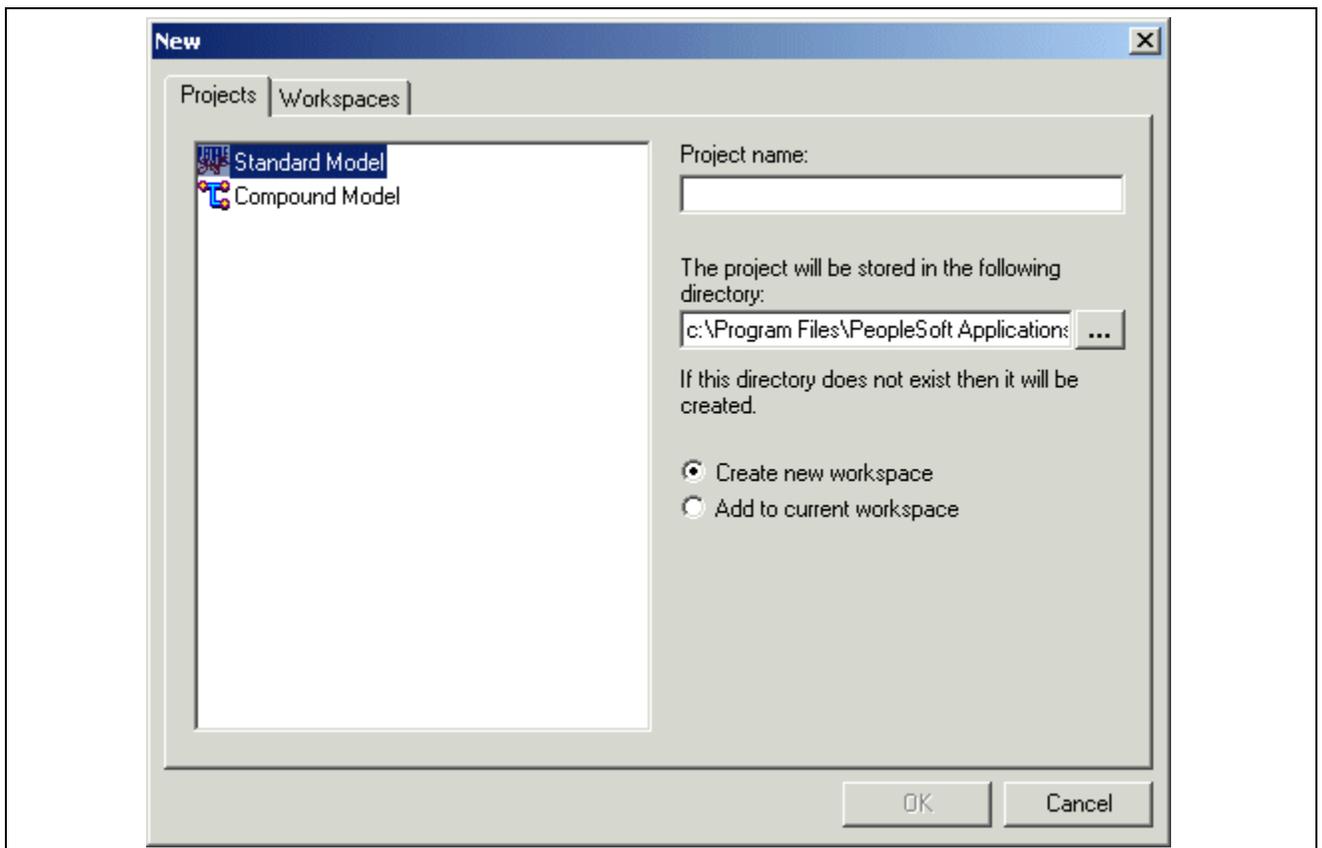
Each Standard model contains a file named standard.cms. This file is included when the project is created. standard.cms contains the root class and must be present in all projects containing standard models.

Team Modeling

The Visual Modeler file structure makes team modeling possible. Using multiple source files, a modeling team can work on the same model simultaneously. Changes can be checked in and checked out using source control. Additionally, most merge tools support the CMS format.

Creating a New Project or Workspace

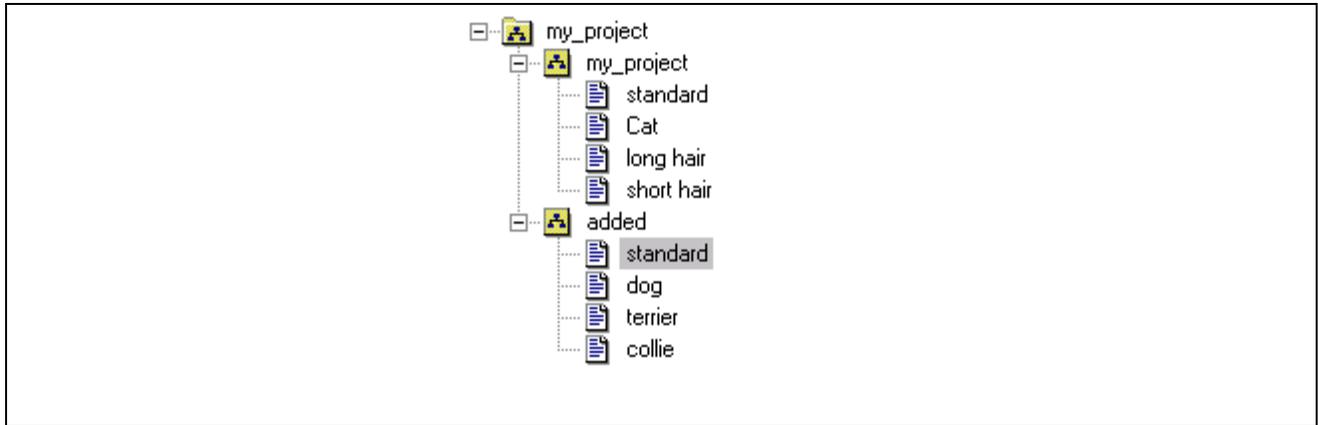
To create a new project in a new workspace, select File, New and click the Projects tab. Select the model type, standard or compound. Click Create new workspace. Browse for or type in a project directory, then type in a Project name; the name will be appended to the project directory path. Click OK.



Creating a new project

You can store multiple projects in a single workspace.

To create a new project in an existing workspace, select File, New and click the Projects tab. *Click Add to current workspace*, then browse for a project directory. Specify the Project Name, then click OK. The new project appears in the specified workspace; it has its own structure, as shown in the File view:



File view

When there are multiple projects in one workspace, you must explicitly set the active project so that Configurator system knows which project to compile. Select Project, Set Active Project to do this.

The File, Save Model As command allows you to save a copy of the model. All of the files in the project will be copied into a new destination directory and all occurrences of the original project name (model name) are replaced with the new project name. Source control status for the project and the source files will be cleared (you will need to add the new project to source control). The new project then becomes the active project.

The File, Save All option saves all model and workspace information for the current project. In addition, a model information file is created in the source workspace path. The filename format is <modelName>.modelinfo.xml. The Configurator Extensions for DreamWeaver use this file.

See [Chapter 28, “Using the Page Editor Extensions for Dreamweaver,” page 359](#).

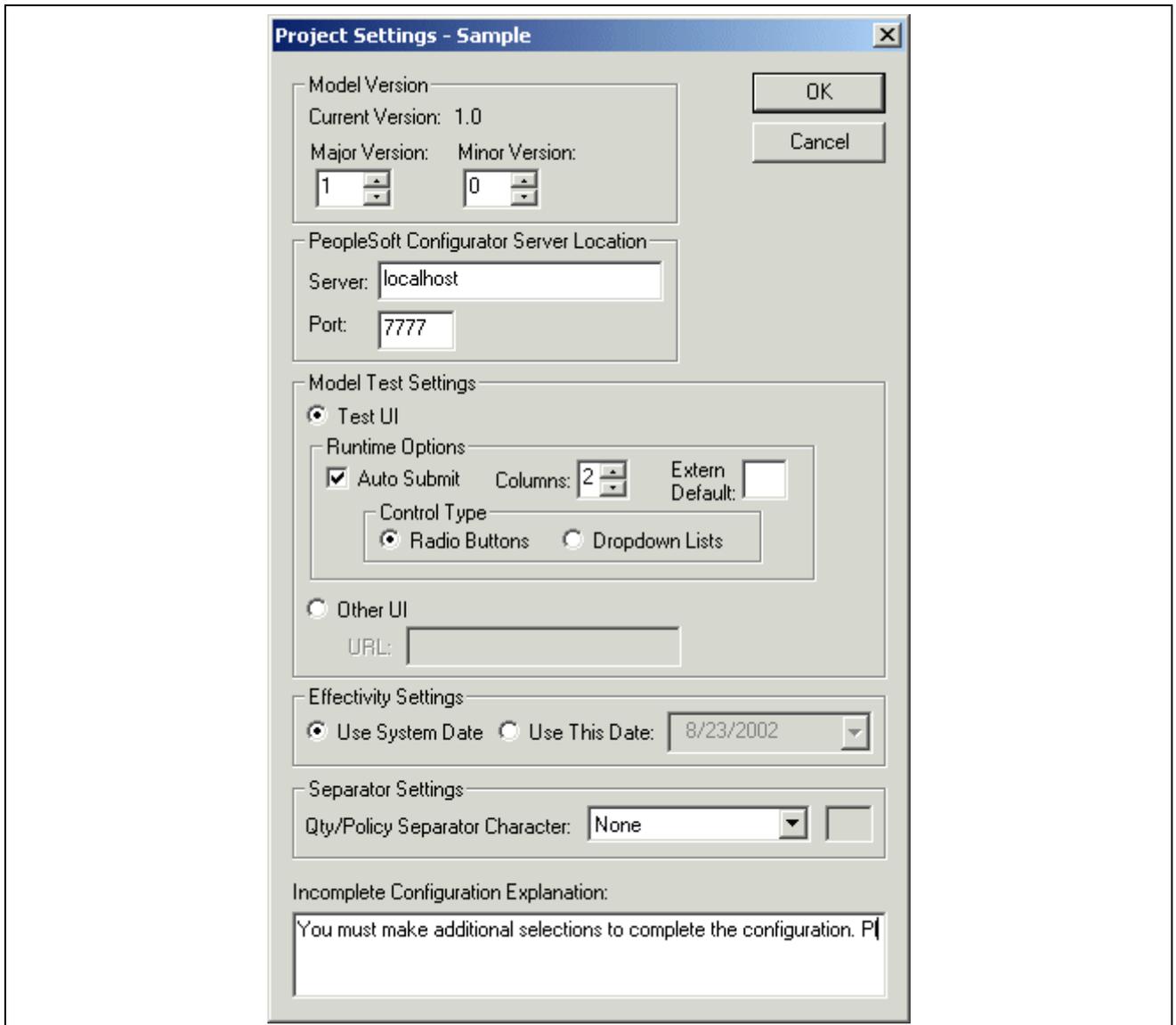
Visual Modeler provides a way to create (save) a description of the model to an HTML file. Select File, HTML Report.

Specifying Model Project Settings

You can specify model-level information in the Project Settings dialog.

Dialog Used to Specify Model Project Settings

Select Project, Settings. In this dialog you can specify compilation options and settings for use at compile time and run time.



Project settings for a component model

Current Version	The number of the current model version before any changes to the Major or Minor Version.
Major Version	An arbitrary model version used for model maintenance in <install directory>\ViM\models.
Minor Version	An arbitrary minor version used for model maintenance in <install directory>\ViM\models.
Server	The server on which WebLogic and Advanced Configurator are installed. For example: <code>my_machine.peoplesoft_config.com</code> .
Port	The port number for the WebLogic application server. It is 7777 by default.
Test UI	Uses the Model Tester provided with the Advanced Configurator installation (component models only).

	<ul style="list-style-type: none"> • <i>Auto Submit</i>—Check this box to turn on Auto Submit in the Model Tester. If Auto Submit is on, the page is refreshed with each pick. • <i>Columns</i>—The Model Tester displays runtime controls using a table form. Specify the number of columns in the output display. • <i>Extern Default</i>—The value to apply to any extern variables at model initiation. • <i>Control Type</i>—Select either of Radio Buttons or Dropdown Lists. This selection applies to all controls.
Other UI	Use your own JSP pages.
Effectivity Settings	Choose test date settings to verify effectivity dates on relationships. <ul style="list-style-type: none"> • <i>Use System Date</i>—Use the local system’s date (usually the current date). • <i>Use This Date</i>—Enter a date in or out of an effectivity range.
Separator Settings	Specify the character you use in SQL queries for separating the Quantity and Policy values.
Incomplete Configuration Explanation	Text to display to the user when a completeness check is run. You can compose the message with these parameters: <ul style="list-style-type: none"> • Selection point name— \$ (n : \$DPNAME) • Expression value— \$ (expr : exprname) <p>If the name parameter is included within the explanation text, a separate explanation message will be generated for each required selection point that does not have a selection. Otherwise, a single explanation message will be generated for the Incomplete Configuration violation.</p>

Adding a Project to Source Control

For a description of the source control interface and instructions on configuring source control options.

See [Chapter 3, “Understanding Modeling,” Source Control Interfaces, page 47](#).

1. To add a project to source control, select one or more components in the Visual Modeler File view.

If the Visual Modeler does not recognize files that are already in source control, check out the files and check them back into the same location. The Visual Modeler will detect the files and duplicates will not be created.

Note. Some providers will always prompt for a log in when a workspace or project file is checked in. Consequently, when adding all the files in the File view, you will be prompted to log in once for the .cms files and again for the project file. This is not an extra message. Be sure to provide the project information; do not leave the field blank. This behavior is known to occur with Visual SourceSafe.

2. Right-click in the tree view, then select Add to Source Control. You will be prompted to add a comment. To retain checked out files, check the “Keep files checked out” option.
3. Click OK to send the source control request to the provider software; at this point the source control software takes over.
4. To check out a file, select it in the File view, right-click, then select Check Out.

The source control status is visually depicted as follows:

	File not in source control.
	File checked in.
	File checked out
	Project or Workspace not in source control.
	Project or Workspace checked in.
	Project or Workspace checked out.

Importing and Exporting Models

Advanced Configurator provides a means to manage, update, and maintain models by importing and exporting model data. If you have model data that you would like to combine with another model—even if it is not currently an Advanced Configurator model—or if you want to update an older model by overwriting all or some of its components, you can use the export and import commands in the Projects menu.

These commands allow you to import and export all or parts of a model:

- Database references.
- Classes, with or without domain members and SQL queries.
- Selection points.
- All or selected relationship types.
- Expressions.

Importable models must be created in one of two ways:

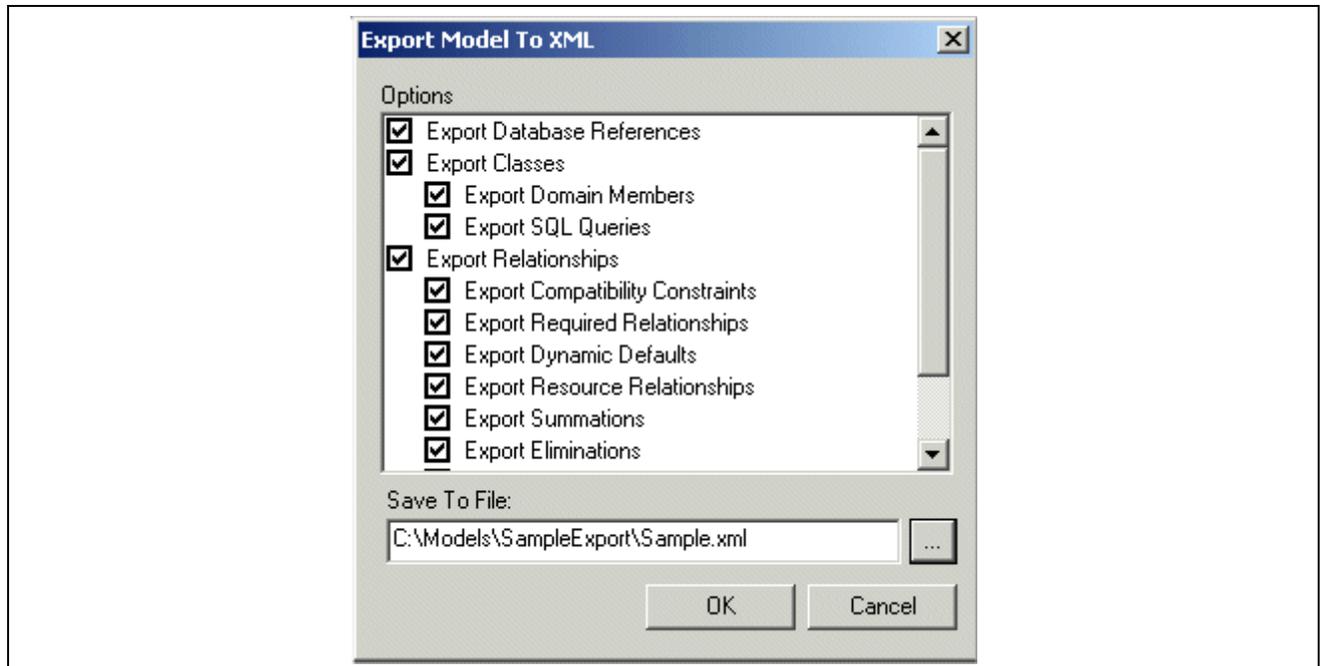
- From an Advanced Configurator model that has been exported, as this formats them in the XML defined by PeopleSoft Configurator Interchange Format (PCIF).
- From an XML file that compiles with the PCIF.dtd.

The latter method allows you to import a model, partial or complete, from an outside data source.

See [Appendix G, “PCIF,” page 477](#).

Exporting a Model

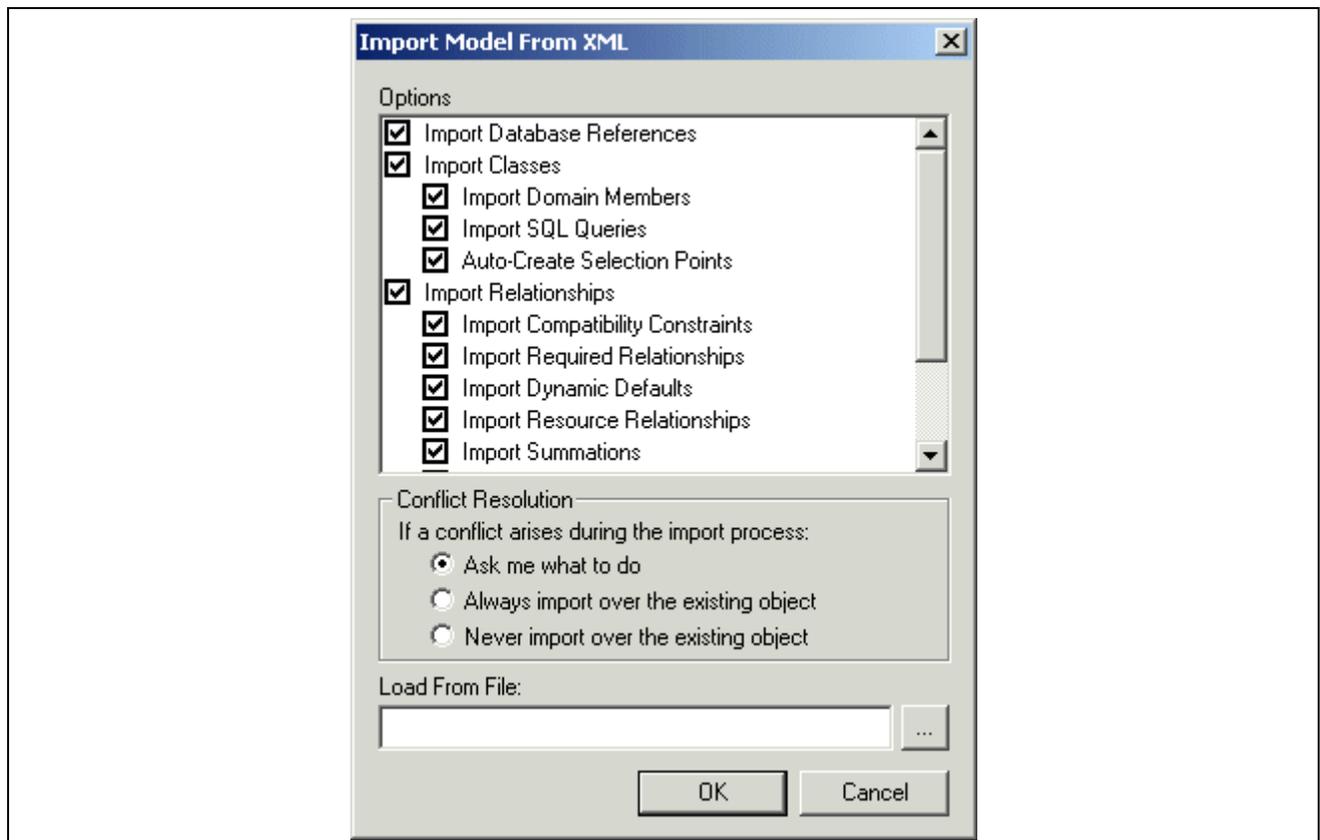
To export a model, open it in the Visual Modeler, and choose Projects, Export Model. The Export Model to XML dialog appears.



Export dialog, showing default settings with all components selected

Importing a Model

The dialog box for importing a model is very much like that for exporting a model.



Import dialog, showing default settings with all components selected

To import an Advanced Configurator model:

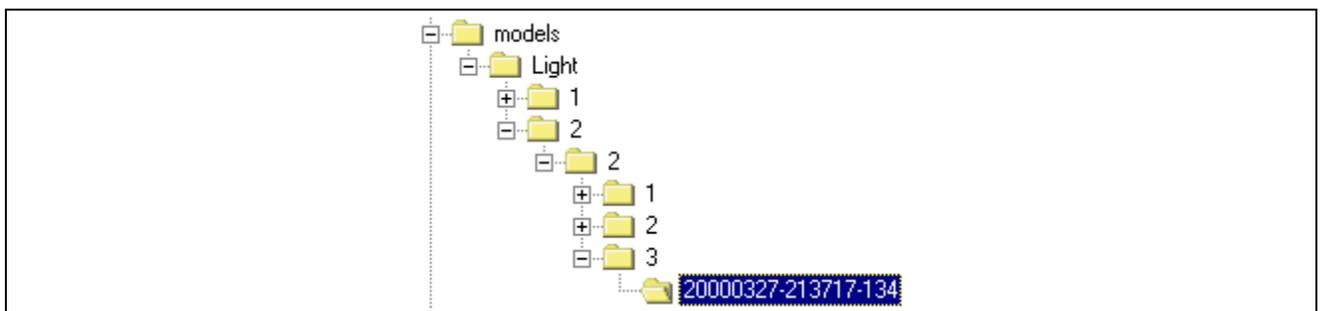
1. Open the target model in the Visual Modeler and select Projects, Import Model.
The Import dialog appears.
2. Uncheck any components that you don't want to appear in the target model.
3. Select the desired method for handling conflicts that result when a component being imported has the same name as an existing component.
These options let you control, on a case-by-case basis, whether components are overwritten or not.
4. Choose the desired filename for the import file (XML) and click OK.

Compiling a Model

Compiling a model triggers several separate actions.

- The current model is compiled.
- At compilation, an intermediate XML file is created in the TEMP environment variable.
- If an intermediate XML file does not yet exist, a directory with the same name as the model is created on the Configurator server in `\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\Web-inf\models`.

This process preserves a hierarchy of files.



A typical hierarchy of model files

In the above hierarchy,

- The directory name Light is derived from the model name.
- The Major versions, 1 and 2, were taken from settings in the Project Settings dialog, as was the Minor version (2.1).
- Any sub-minor versions (1, 2, and 3) are created automatically whenever the model structure changes or the Explanations information changes.
- Each compile is stored in its own directory identified by the compile ID. In the above sample, this is 20000327-0333825-244. The ID is extrapolated from the date and time. A sub-minor version can have many compiles.

Advanced Configurator provides a Model Tester for component models. It is a JSP page for testing model relationships. The Configurator engine must be running in order for the compile to take place.

Note. Compilation takes place for the Active Project, which may not necessarily be the project you are currently viewing in the Visual Modeler. To check the active project, select Project, Set Active Project.

Advanced Configurator provides tools for maintaining the contents of the \models directory and the temporary files in \TEMP.

See Chapter 35, “Maintaining the Advanced Configurator System,” page 425.

To compile a model:

1. Start the Configurator server by either running `<WebLogic home>startConfigurator.cmd`, or selecting Start, Programs, PeopleSoft Applications, Configurator 9.0, Start Configurator Server.
2. Click the Compile and Run icon on the toolbar to compile the model and launch the Model Tester. If you would like to save before compiling, select Tool, Options, then select the Save before Run option on the General tab.

Alternatively, click the Compile icon on the toolbar to compile the file without launching the Model Tester.

3. Check the output window (below the Model Structure View by default) for compile-time messages, warnings, or errors.

At least one relationship must be specified in order to compile a model. In addition, each class that participates in a relationship must have a selection point. (Click the Selection Point icon on the toolbar to display the selection points in the Model Structure View.)

Using the Model Tester

The Model Tester displays the following information and display options.

The screenshot shows the Model Tester interface with three main sections:

- Model Info:**
 - Model Name: Sample
 - Model Version: 1-0-18
 - Compile ID: 20030718-143415-294
- Legend:**
 - Eliminated (X icon)
 - Conflict (Warning triangle icon)
 - User Selected (Hand icon)
 - Computer Selected (Computer icon)
- Options:**
 - Show Elimination Level
 - Auto Submission Of Picks
 - Verify Configuration
 - Use Select (List) Controls
 - Sort By State
 - Show None On Required Controls
 - Number Of Controls Per Row:
 - Attribute To Display:
 - Solve date (YYYYMMDD):

Model Tester information and display options

Model Name

The name of the model as it appears in the Visual Modeler project.

Model Version

Displays the model’s version. The major and minor versions can be set in the Project Settings dialog.

Compile ID

The compile ID is based on the time the file was compiled on the application server.

Show Elimination Level	If the model contains elimination levels, display them on the Model Tester.
Verify Configuration	If Verify Configuration is checked, the Advanced Configurator checks the current set of picks for missing selections (on required controls) and violations. The model is assumed complete and valid if all required selections are made and no violations are detected.
Sort By State	Display the most recent pick at the top of the control.
Auto Submission of Picks	Refresh the page as soon as a pick is made. If Auto submission is off, make one or more picks, then click the Submit button. This value can be set from the Project Settings dialog.
Use Select (List) Controls	<p>If this box is not checked, radio button controls will be used for the entire model. If this box is checked, list control drop downs will be used. When list controls are selected, text indicators, rather than the graphics shown in the Legend box, are displayed:</p> <ul style="list-style-type: none"> • D_S—Default selected (used when None is selected by default) • C_S—Computer-selected • C_E—Computer-eliminated • U_S—User-selected • U_E—User-eliminated
Show None on Required Controls	None is added to single-select controls at runtime so that if List controls are used the control can initialize with nothing picked. You can hide these values on single-select controls.
Number of Controls Per Row	The number of columns in the Model Tester table. This value can be set from the Project Settings dialog.
Attribute to Display	Display an attribute that is defined for the domain member instead of the Name. May also display an attribute that is not defined in the model, but is part of the class. For example, a long description could be displayed even though the model knows nothing of that description.
Solve Date (YYYYMMDD)	<p>Displays the date that the Model Tester is using to test the model. If the box is initially blank, the system date is being used. If a date appears in the box initially, it is the specified Model Tester solve date for the project. (The model test solve date is set in the Projects Settings dialog.)</p> <p>You can specify a different date in the Model Tester and re-run the test under the new date by clicking the Submit button. The model is not recompiled.</p>

Note. Be aware that, in the Model Tester, four or more controls placed horizontally may not be displayed properly in Microsoft Internet Explorer.

Internalizing Model Data

PeopleSoft Enterprise Mobile Configurator allows the end-user to configure products with Advanced Configurator applications on local, mobile environments common to field sales. Being local, such environments do not have access to corporate databases containing model data. To eliminate this limitation, Advanced Configurator allows system administrators to acquire the data and store it in the model before distributing the model to the field users.

Internalizing the data can be accomplished in the Configurator Administration Console or with the Visual Modeler. If you have auxiliary files that must accompany the data, use the Administration Console. Also, internalizing data in the Visual Modeler is a one-time operation that will create a model with internal data. Internalizing data with the Administration Console preserves the SQL query information in the model, yet still produces a compiled model that doesn't need the database to run. If the database data changes you can re-compile to pick up the new data.

To internalize model data using the Visual Modeler:

1. Make a copy of the model's files (in Explorer).

Note. You must internalize a copy so that there remains a database-connected model for continued maintenance. You will always make a copy of the latest version to use in creating a distributable Mobile Solution Package.

2. Open the copy of the model in the Visual Modeler.
3. Select Project, Internalize Model.
4. Select *Yes* to internalize data.

Internalization automatically converts model data into domain members and constraints. If the data is very large, this process may be lengthy.

CHAPTER 5

Creating Objects for the Model

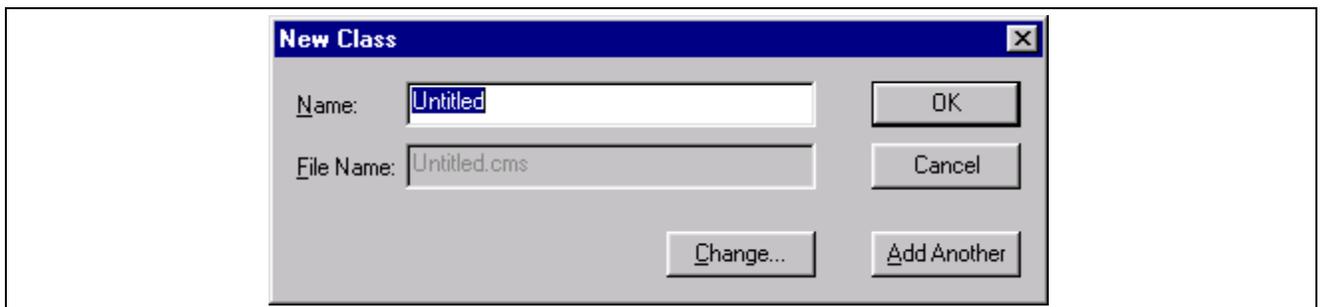
This chapter discusses how to:

- Create a class.
- Delete a class.
- Change class structure.
- Add class attributes.
- Create internal domain members.
- Create external domain members.
- Create a “None” domain member.
- Assign values to attributes.
- Inputting date-type attributes manually.
- Set up binding for external domain members.
- Filter and manipulating table data.
- Work with selection points.
- Storing a dynamic default quantity in a database.
- Internalize data.

Creating a Class

A class has the following default properties:

Inheritance principles must be considered while a model is being built. Click anywhere in the model structure view to ensure focus on the modeling area.



New Class dialog box

To create a class:

1. Click the Create Class icon in the right margin, then click the parent class, or, select a class, then select Insert, Class.

To create multiple classes, hold down the Shift key, click the Create Class icon, then click the parent class, or, select a class, then select Insert, Multiple Classes.

The New Class dialog appears.

Note. A class may have either subclasses or domain members. If a class has domain members, they must be deleted before a subclass can be added.

2. Type a name into the Name entry field of the New Class dialog. The dialog prompts with a .cms file name based on the string you enter. Before continuing be sure the name is correct according to the following:
 - Class names must be unique within a model.
 - The name cannot include \, =, <, >, :, ", (, or). The initial character cannot be dollar (\$) or underscore (_), but these characters can be included in other positions.
3. Click the Change button to specify a different file name. A class object can be saved to an existing .cms file.

Once a name is accepted, the class name can be changed within the model, but the file name cannot change.
4. If you are adding a single class, click OK to end the process, or click Add Another to continue adding classes. Or, If you are adding multiple classes, type "Return" to make another, or click the cursor arrow in the menu bar to revert to the normal Windows cursor.

When a class is created the following events occur:

- The information for the new class is displayed in the context-sensitive properties editor (if it has focus).
- The class name appears in the Class list in the Components view.
- The file name appears in the file list in the File view.
- The file name is automatically added to the model's .csp file.
- If the class is a leaf class, a corresponding selection point is automatically created.

Deleting a Class

To delete a class, select it and press the delete key.

Deleting a class:

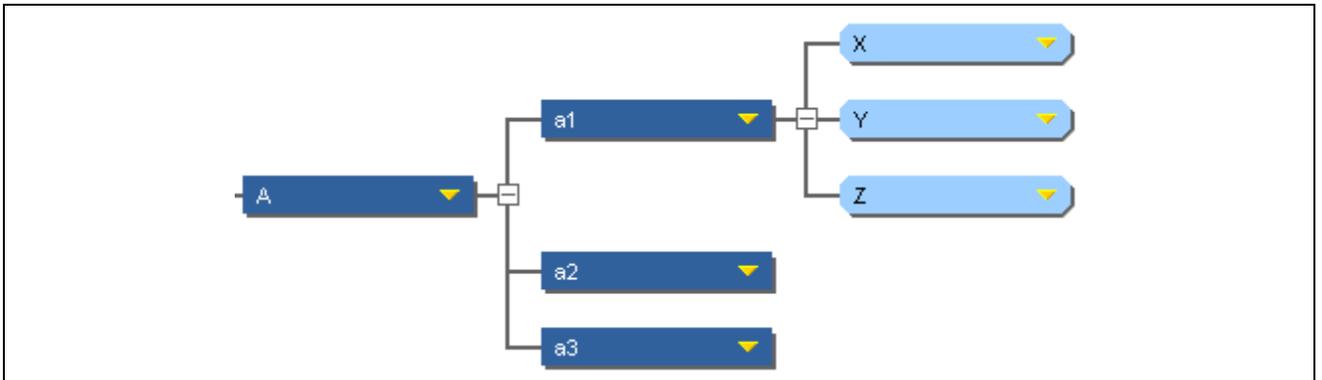
- Removes the class, any subclass(es), and any domain members.
- Removes the class from any relationships that reference it.

To remove the corresponding file from the .csp file, click the File View tab. Locate the associated class file and right-click to open the context-sensitive menu, then select *Remove File From Project*. Note that deleting a class in Visual Modeler does not remove the corresponding .cms file from the file system.

Changing Class Structure

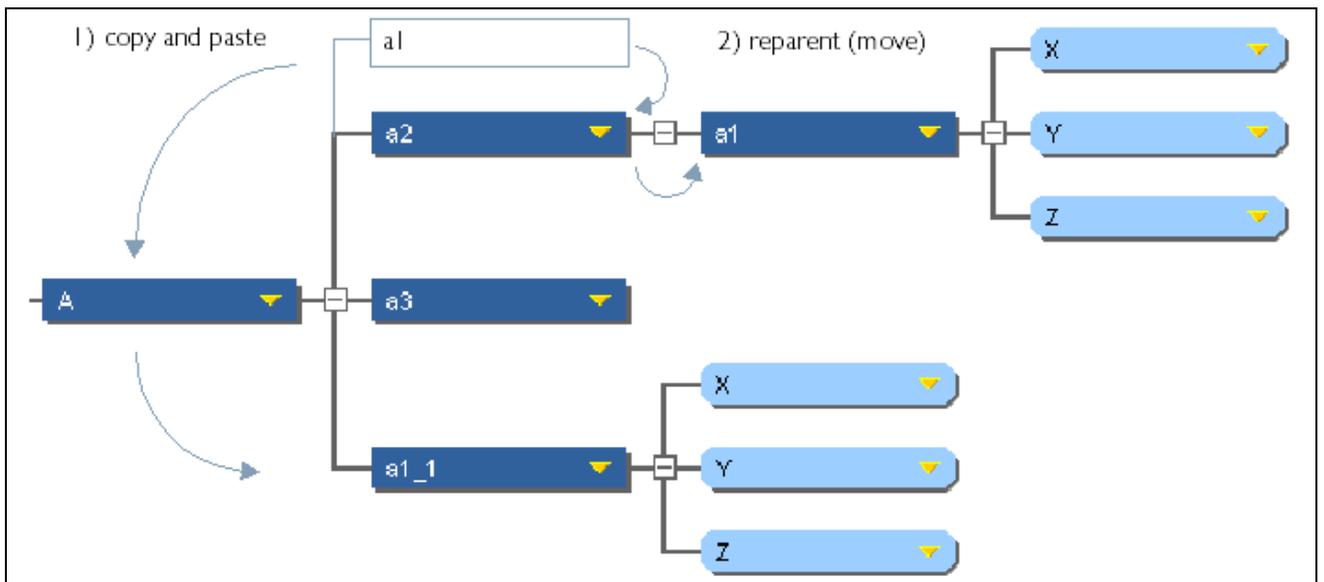
It is possible to reuse parts of a diagram with the Reparent and Copy and Paste features.

Compare the following two figures:



Original structure

- a1 is copied and dropped on the Root class. Because the new class is a sibling of a1, a new name, a1_1 is automatically created.
- The original a1 is dragged and dropped onto a2. a1 becomes a subclass of a2; there is no need for a name change.



Copy and paste, and reparent

To reparent a class:

Drag and drop it on a class that does not have domain members.

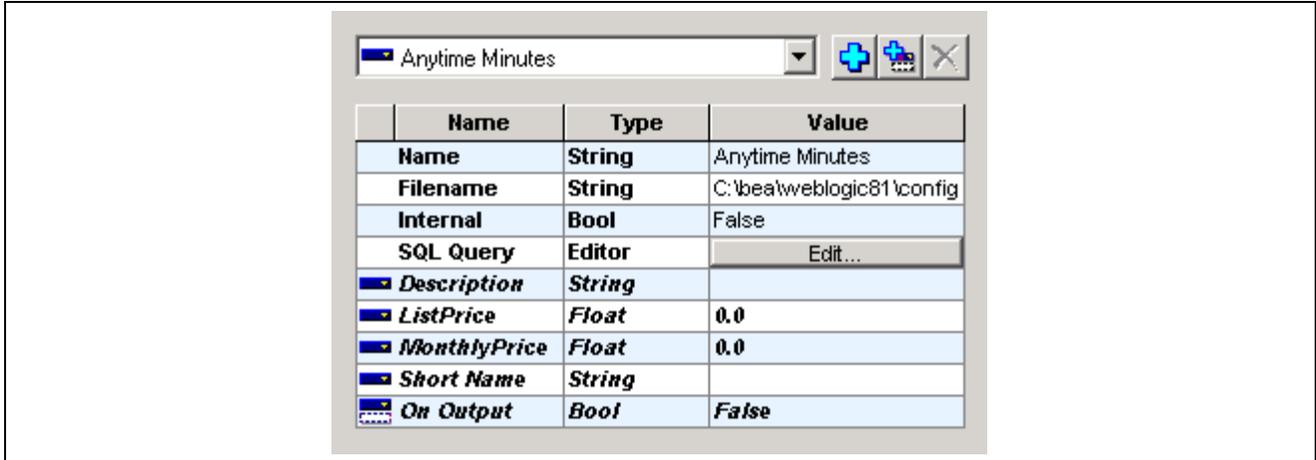
To copy and paste a class:

Copy a class and paste it on another class that does not have domain members. The original class remains unchanged in the diagram. Any domain members and relationships will be transferred along with the copied class to its new location. If the class is on the same hierarchical level (a sibling) the class will be automatically renamed to keep it unique. If the class is not a sibling, the name is unchanged.

Note. Copying and pasting a class will not copy the corresponding selection points. If the copied class is a leaf class, a selection point will be created for it; however that selection point will have the default properties, not the properties, of the original class's selection point.

Adding Class Attributes

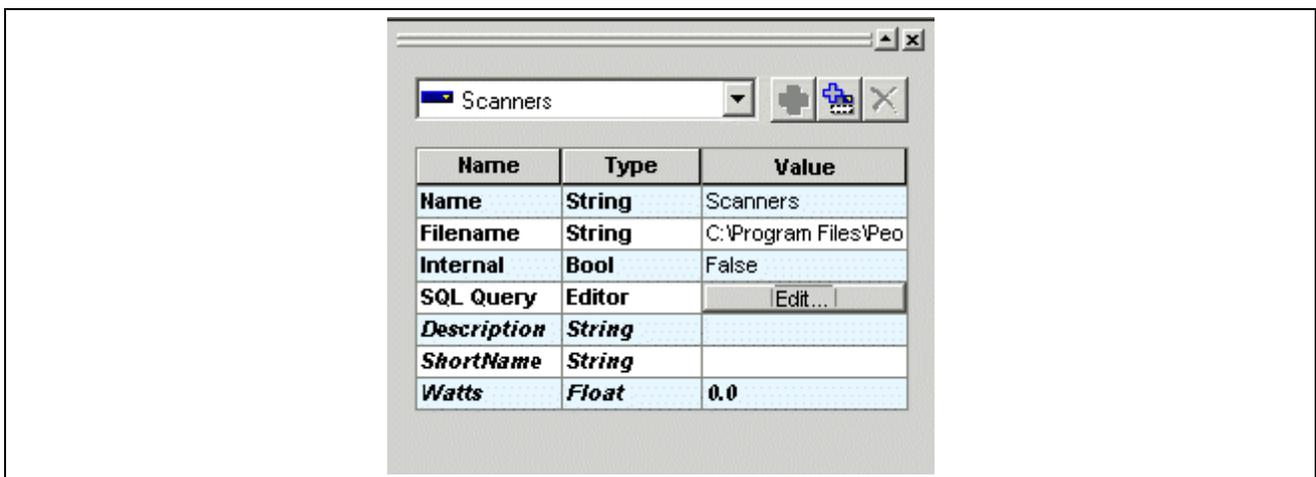
Class, domain member, constraint, and selection point values and attributes can be specified or edited in the context-sensitive Properties editor.



The Properties editor for a selection point

The figure shows the Properties editor for a class in which the user has defined attributes, and the Internal property is set to False, so that the attribute values will be imported from an external source.

Note. The SQL Query field is not shown unless Internal is set to False.



Class or subclass properties

To add an attribute to a class:

Click the Add button at the top of the panel to open the Create a New Property dialog. You can either use the dialog to enter the name, type, and value, or click the OK button to add a blank row to the table for editing. Possible types are Boolean, float, int, String, and date.

An entry in the Value column becomes the default for child classes and for domain members. The value can be changed in the class in which it is defined, or in an internal domain member. When the SQL Query property is set to False, internal values are generally ignored, however, if a database happens to have a null value, the default value from the model will be substituted.

Creating Internal Domain Members

A domain member is an instance of a class. You can create any number of domain members. An internal domain member is one that is stored within the model rather than a database or another system file.

To create internal domain members:

1. To create a single domain member, click the domain member icon, then click the parent class.
To create multiple domain members, hold down the Shift key, and click the domain member icon, then click the parent class. Alternatively, select a class, then select Insert, Domain Member or Insert, Multiple Domain Members.
2. Click OK to complete the process, or click Add Another to continue adding domain members on the same class. The New Domain Member dialog appears.
3. Type in a name. The name cannot include \, =, <, >, :, ", (, or). The initial character cannot be dollar (\$) or underscore (_), but these characters can be included in other positions. An asterisk (*) cannot be used alone, but it can be used in combination with other characters.
4. When you are finished creating domain members for this class, select another class, or click Cancel.

Or,

To stop creating multiple domain members, click the cursor icon in the Windows menu bar to revert to the normal Windows cursor.

Creating a “None” Domain Member

When a control is single-select, an item named *None* is automatically displayed in the Model Tester when you launch it. If the selection point flag Optional is set to True, *None* is computer-selected (unless a default was set). It is important to understand that this “generated” None is not a domain member, and it has no value. As such, *None* does not participate in constraints.

When a user selects *None at run time*, it means “there is no selection made on this selection point.”

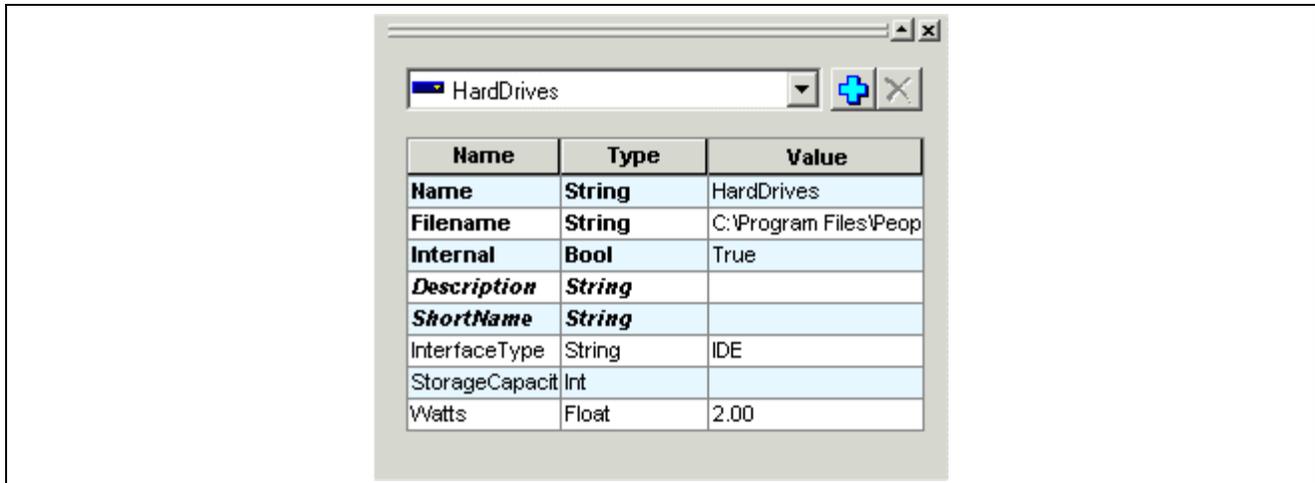
None is not generated for a multi-select control.

For required single-select controls, none has no meaning, so the Model Tester has an option to hide None on required controls.

If you want an item named *None* to be selectable on a multi-select or single-select control, you must create a domain member for that purpose. It is helpful to name this domain member something other than None, such as “No Thanks” or “I don’t want any” to avoid confusion with the “generated” None.

Assigning Values to Attributes

Type appropriate values in the Value column. The figure shows the attributes for an internally defined domain member. The naming restrictions for domain members also apply to attributes.



Domain Member Attributes

Note the use of font styles in the Properties editor:

- Items that cannot be changed are shown in black bold face. For example, the Name and File Name types cannot be changed.
- Items shown in plain text can be modified, that is, the value of the name can be changed.
- Items defined in an ancestor class—for example, the attribute price in the figure, are shown in italic font.

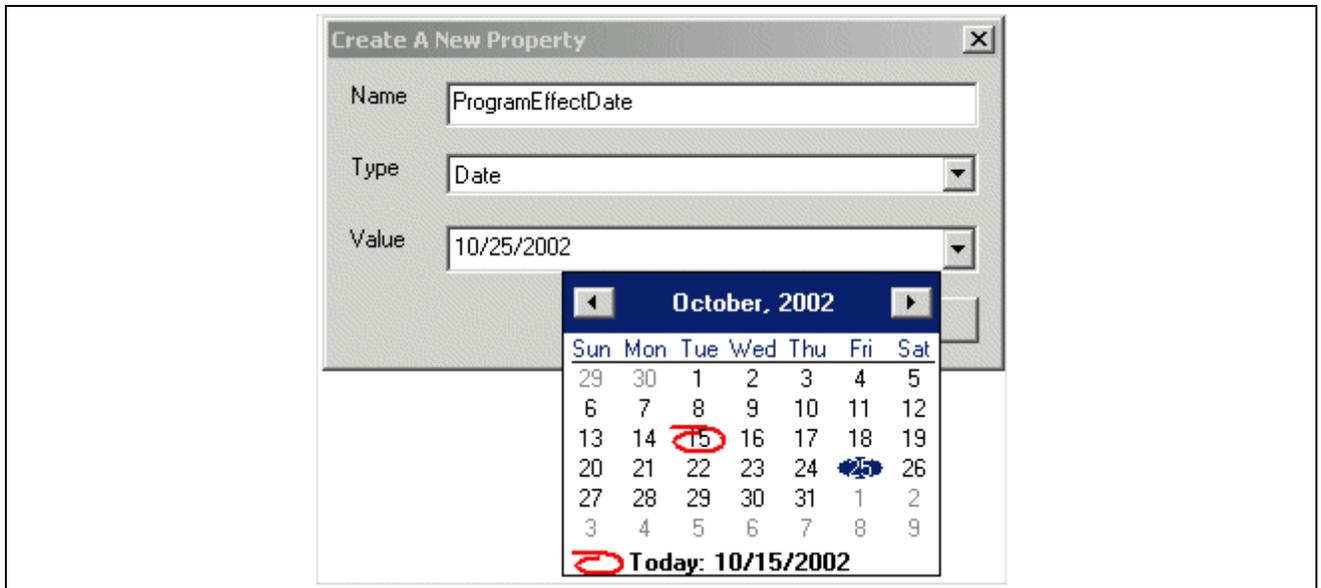
Inputting Date-Type Attributes Manually

You can input date-typed attributes on classes and domain members manually, through the Property Editor, and through an SQL query from a database. The Visual Modeler also supports date-typed data in compatibility, requirement, and dynamic default constraints.

You can express values for dates using the full ISO 8601 format. However, any time zone, hour, minute, and sub-second information will be truncated during the compilation process to yield a YYYY-MM-DD format. The validity of dates is checked during the compilation process and error messages are generated for any invalid dates detected. Advanced Configurator stores the dates as strings in their YYYY-MM-DD format since many date operations can be performed on the ISO 8601 format using string manipulation and comparison functions.

Date values loaded from a database are retrieved and converted to the YYYY-MM-DD format. Thus, you can store and retrieve date values in formats other than YYYY-MM-DD.

Note. Strings can be input directly from the user interface if they are in a parseable date format.



Editing a class property with date type

You can also use date data in expressions.

See [Chapter 6, “Creating Relationships Between Model Objects,” Creating and Editing Expressions, page 89.](#)

Setting Up Binding for External Domain Members

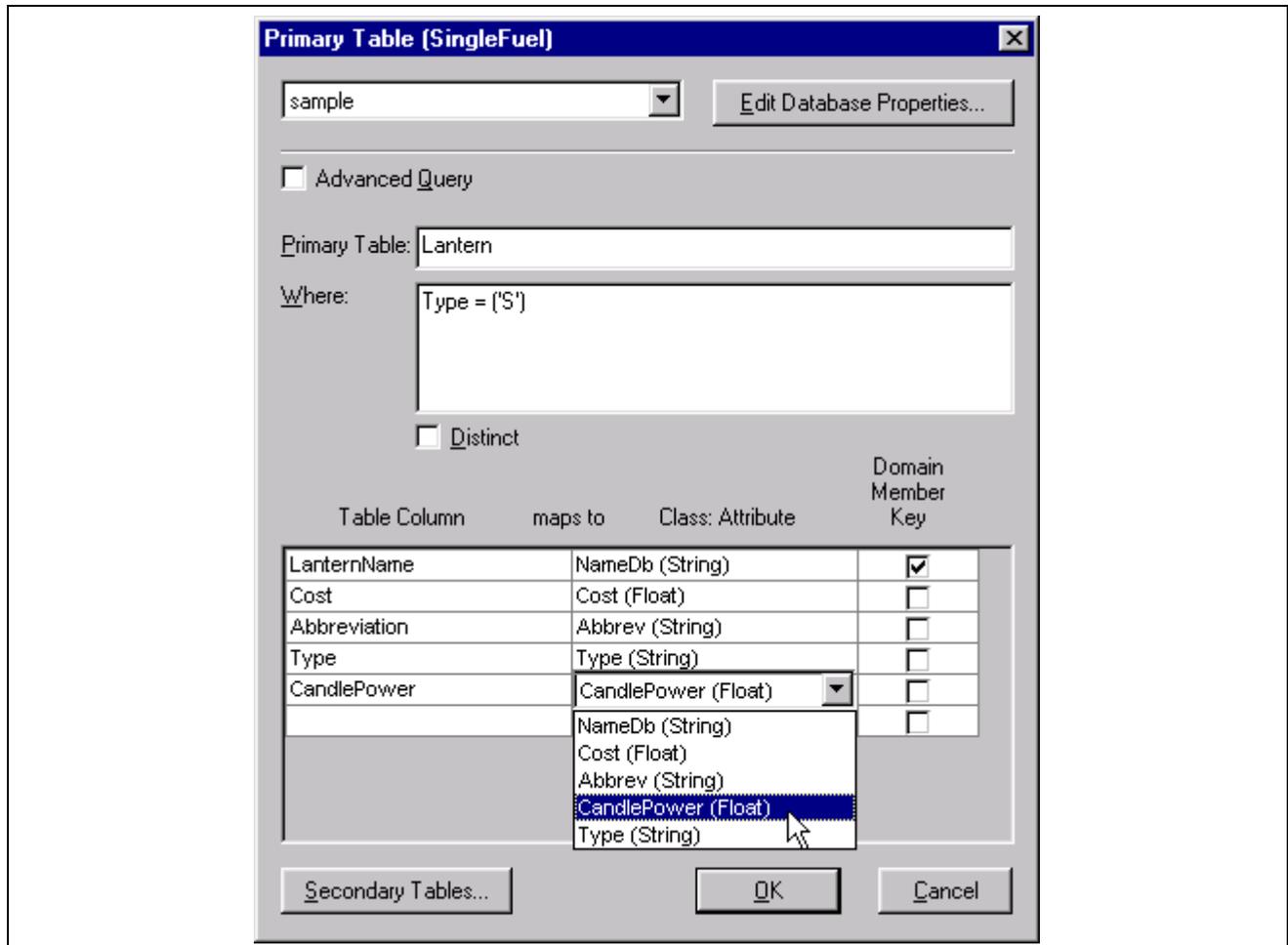
Relationships can be created between classes or selection points and domain members obtained from a database. Setting up the modeling environment to work with external objects requires:

- Configuring the database.
See [Chapter 4, “Setting Up the Modeling Environment,” Database Interface Configuration, page 51.](#)
- Configuring your system so that you can see external data in the model.
See [Chapter 4, “Setting Up the Modeling Environment,” Connecting to a Database from Visual Modeler, page 54.](#)
- Specifying the appropriate values for properties in the JNDIDBName.properties file so that you can compile and run a model with external data.
See [Chapter 4, “Setting Up the Modeling Environment,” Configuring JNDIDBName.properties, page 53.](#)

Selecting a Primary Table

Select a class, and in the Properties editor, set Internal to *False*. The SQL Query field appears. Click the Edit button to display the primary table dialog. (If no databases are listed in the drop down on the upper left, the Visual Modeler is not connected to a database.)

See [Chapter 4, “Setting Up the Modeling Environment,” Specify a Database Connection, page 54.](#)



Primary table with attribute mapping and WHERE clause

Simple Queries

Simple queries, where a database table column corresponds exactly to the model needs, do not require an advanced SQL query.

To write a simple query:

1. Type the database table name into the Primary Table field.

Note. The table name and all other names associated with the database are case-sensitive.

2. Type a Column name from your database into an empty Table Column cell. To map the values to class attributes, make a selection in the corresponding Class:Attribute drop down.
3. (Optional) Enter a Where clause. The SELECT and FROM portions of the query are derived from the dialog inputs. You need only enter the WHERE portion.
4. (Optional) Check the Distinct check box to ensure that repeated values are not displayed.
5. Click the Domain Member Key box next to the value you want displayed on the domain member. At least one row must be chosen as Domain Member Key. The Domain member key does not have to match the Primary Key in the database.

Note the following important points about queried data:

- Data queried for a Domain Member key must be unique. The Domain Member key data will be used as an identifier for domain members at run time.
- If two domain member keys are chosen, both will be displayed on the imported domain members. The topmost key will be displayed first.

If more than one domain member key is chosen, the union of all keys will be used to identify domain members at run time; the combined list cannot have duplicates.

- Queried data has the same restrictions as internal data. A query cannot return a name that includes \, =, <, >, :, ", (, or).
The initial character cannot be dollar (\$) or underscore (_), but these characters can be included in other positions. An asterisk (*) cannot be used alone, but it can be used in combination with other characters.
- *None* is displayed on single-select controls at run time. This “generated” None is not a domain member. If you want a domain member to perform this function, you must create one in the database.

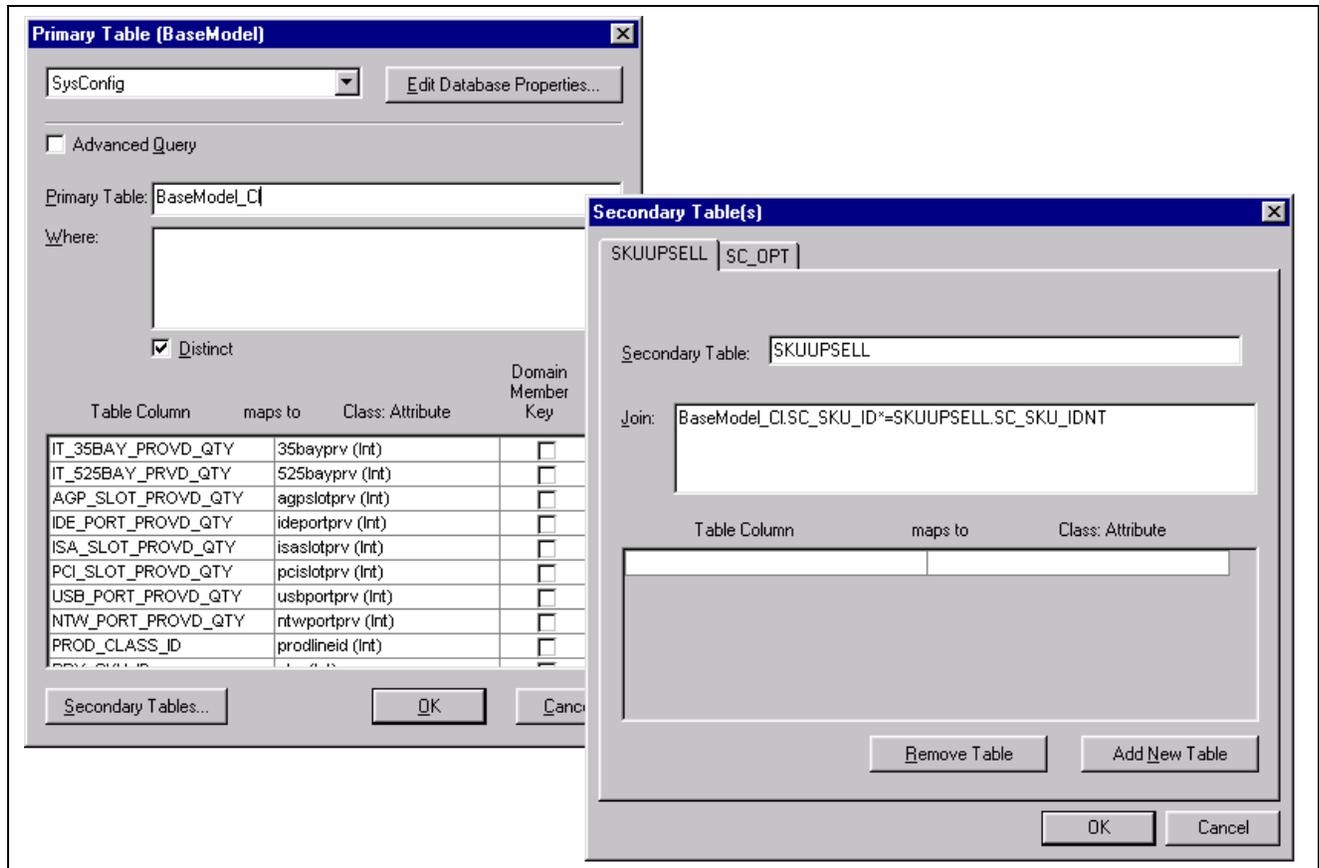
See [Chapter 5, “Creating Objects for the Model,” Creating a “None” Domain Member, page 73](#).

Secondary Table Option

To join a secondary table to a Primary table, click the Secondary Table button. Database-to-class property mapping is similar to that for the Primary window, with these exceptions:

- The SQL query is a Join clause. Typically, a secondary table is used to add an attribute from another table to an external domain member. The secondary table has at least one column in common with the primary table. The common column is used to state the join condition.
- Columns cannot be mapped to domain members in this dialog.
- Multiple secondary tables are allowed. Click Add New Table. Each table appears on a separate tab in the Secondary Table(s) window.

Note. Limit using secondary tables to situations where they are truly necessary. Secondary tables are preferred to defining advanced queries, but they slow Visual Modeler performance and increase compilation time.



Primary and Secondary Tables with SQL Queries

Advanced Queries

An Advanced Query is required if filtering must be applied to the table data and if you plan to do any data manipulation in the SQL. For example, you could calculate someone’s age based on the current date and their birth date and return that value as AGE. You may also need an advanced query if you are using nested select statements or complex joins.

When you use an advanced query, the runtime system can not dynamically read additional database data as requested by run time components. Any data needed must be provided by the query specified in the Advanced Query specification.

You can not request additional attributes on the UI with advanced queries. For example, with a simple query, you don’t have to map LONG_DESCRIPTION into the model, but you can still display it at run time since it is added onto the query for the class. With advanced queries, all attributes that will be displayed or otherwise referenced (e.g., in pricing) must be mapped into the model.

Note. An advanced query must be written such that the first column returned is returned as “\$NAME”. This should be the domain member key (or equal to the domain member key).

To enter an advanced query:

1. Click the Advanced Query check box. The Primary Table field will disappear.
2. Type an SQL Server or Oracle query into the Enter Advanced Query field to:
 - Generate the domain-member name column from the key columns.

- Return it as column 1 and call it '\$NAME'.

This part of the query looks like:

```
(tbl1.keycol1 || tbl1.keycol2 || tbl2.keycol1) AS '$NAME', tbl1.col2 WHERE ...
```

Examples

SQL-Server:

(single-column key)

```
SELECT (cast( tbl1.keycol1 as nvarchar(4000))) AS '$NAME', tbl1.col2 WHERE ...
```

(multi-column key):

```
SELECT (cast( tbl1.keycol1 as nvarchar(4000)) + '_' + cast( tbl1.keycol2 as nvarchar(4000)) + '_' + cast( tbl2.keycol1 as nvarchar(4000))) AS '$NAME', =>
tbl1.col2 WHERE ...
```

Oracle:

(single-column key):

```
SELECT (tbl1.keycol1) AS "$NAME", tbl1.col2 WHERE ...
```

(multi-column key):

```
SELECT (tbl1.keycol1 || '_' || tbl1.keycol2 || '_' || tbl2.keycol1) AS "$NAME", =>
tbl1.col2 WHERE ...
```

(You can also leave out the table prefix (“tbl1.”, “tbl2.”) if there is only one table being queried.)

3. Add the data call portion of the query:

In an example where a class' attributes name, *attr1*, and *attr2* are taken from tableX, and there is a need to order by *attr1*, the query (for Oracle) is similar to this:

```
select name as "$NAME", attr1, attr2 from tableX order by attr1
```

The column mappings are:

```
$NAME -> tableXID(domain member key)
```

```
attr1 -> columnA
```

```
attr2 -> columnB
```

4. Perform the remaining steps as described in the preceding instructions “Simple Queries.”

Note. Do not use the Advanced query unless it is absolutely necessary. The simple query is preferred because it can be optimized at run time.

Using Table Aliases

Using a table alias helps allows you to:

- Refer to the same table/view more than once in a single query.
- Refer to a table by a sorter name in the where clause.

In particular, a table alias can help you create joins and differentiate between multiple instances of the same table. To create a table alias, follow the table/view name with a name of your choosing—the alias. When you refer to that table instance in the where clause, prefix the column name with the prefix. The SQL below is what the system actually generates at run time. Note the prefix names used in the WHERE section—you can see them defined in the FROM section.

```

SELECT DISTINCT ( CAST(kit.PROD_COMPONENT_ID AS NVARCHAR(4000)) ) AS '$NAME'
FROM   PS_PRODKIT_COMPS kit, PS_EOEP_PRICE_LIST list, PS_EOEP_PRICE_LIST recur,
      =>
      PS_PROD_ITEM prod
WHERE  kit.SETID = 'COM01'
      and kit.PRODUCT_ID = 'TEL200002'
      AND list.RECURRING_FLG = 'N'
      and list.PRICE_LIST_ID = 'COM_PRICE'
      and list.SETID = kit.SETID
      and list.PRODUCT_ID = kit.PROD_COMPONENT_ID
      AND recur.RECURRING_FLG = 'Y'
      and recur.PRICE_LIST_ID = 'COM_PRICE'
      and recur.SETID = kit.SETID
      and recur.PRODUCT_ID = kit.PROD_COMPONENT_ID
      AND prod.SETID = kit.SETID
      and prod.PRODUCT_ID = kit.PROD_COMPONENT_ID

```

Filtering and Manipulating Table Data

An Advanced Query is required if filtering must be applied to the table data and if you plan to do any data manipulation in the SQL. For example, you could calculate someone's age based on the current date and their birth date and return that value as AGE. You may also need an advanced query if you are using nested select statements or complex joins.

Storing a Dynamic Default Quantity in a Database

A special character is used to separate the quantity values and policy from the constraint value. For instance, if you want to default four fans with a SUM policy, the string "FANA~4~SUM" might appear in the database to denote that default. The format for a value stored in a database can be one of the following:

```

<value>
<value><delimiter><quantity-or-expression>
<value><delimiter><quantity-or-expression><delimiter><policy>

```

The <value> token is the value to be used by the constraint. In the prior example, this would be the symbol FanA. The <delimiter> token is the special separation character. In the example, this was the character '~'. You can specify the delimiter character in the Project, Settings dialog.

The <quantity-or-expression> token is either a numeric quantity or the name of an expression. The <policy> token is SUM, MIN OF, or MAX OF.

Retrieving Expression Values and Externs from a Database

You can populate LHS expressions and RHS extern values from stored data. Visual Modeler and the compiler recognize data output from the database in specific format and will populate the rows of constraints with that data.

The format for LHS expressions is:

```
<comparator><space><"const" or "f(x)"><space><value or expression name>
```

Examples: `<= const 2` and `> f(x) expSum`

Format for RHS externs is `< "true" or "false">` to indicate whether it is required (true) or optional (false).

You should have one column in the database for each expression or extern in the constraint.

Working with Selection Points

The selection point is the model component that communicates directly with the HTML control in the UI. By default, a selection point is created for every leaf class that participates in a relationship.

A selection point has these properties, which are displayed and editable in the properties table:

Name	Type String. By default, the name of the selection point is the class name with Selection appended. This name can be changed. If you change the name, all constraints or expression that use it must be updated.
Filename	Type String. Filename and path to this object.
Type	Type String. The name of the original class.
Quantity	Type Boolean. Determines default quantities for selections. <i>False</i> indicates that domain members, when selected, have a quantity of 1. <i>True</i> indicates that domain members, when selected, have the quantity of 1 or greater, as set in the Defaults editor (explanation follows), which replaces the Def Choice property.
Use Min/Max	Type boolean. Settings for minimum and maximum number of choices allowed for the control, and for minimum and maximum quantity of each domain member allowed. Determines single- and multi-select control type. When set to <i>False</i> (default), the control is optional, and only one domain member can be selected (single-select). (You can also specify single-selection with the property Multi Sel = <i>False</i> .) When Use Min/Max is set to <i>True</i> , the Defaults editor replaces the Def Choice property
Defaults	The default quantity policies and values for each domain member in the selection point. Click Edit to open the Edit Default Choices dialog. This quantity is compared to the quantity that is assigned to the domain member during run time when it is selected by a dynamic default constraint. It is applied if the dynamic default quantity does not meet the requirement indicated by the policy.

See [Chapter 3, "Understanding Modeling," Quantities in Modeling, page 32](#).

Quantity Policy column—One of four ways to apply the static quantity value in the Quantity Value/Expression column to each domain member:

- *Overridable, Overridable $f(x)$*

Assign the quantity to the domain member only in the absence of a dynamic default quantity on the domain member.

- *Min of, Min of $f(x)$*

Check that the largest of the dynamic quantities assigned to the domain member (assuming it was selected more than once) is the specified static quantity or greater. If it is not, assign the static quantity value.

- *Max of, Max of $f(x)$*

Check that the largest of the dynamic quantities assigned to the domain member (assuming it was selected more than once) is the specified quantity or less. If it is not, assign the static quantity value.

- *Sum, Sum $f(x)$*

Check that the specified quantity equals the sum of the dynamic quantities assigned to the domain member (should it be selected multiple times), otherwise assign the static quantity.

Quantity Value/Expression column—Static quantity. Click on the cell to display the selector arrows and set the static quantity value, or a predefined expression, of a domain member. If you use an expression, the value resulting from the expression will be used.

A value of 0 indicates that there is no minimum, maximum, or summed quantity requirement on the domain member. $1 - n$ is the quantity to be compared to the dynamic quantity, to be applied to the domain member if the policy requirement is not met.

Domain Member column—(not editable) Name of the domain member in the selection point.

Use Quantity Policy for All Domain Members—Specifies whether and how to apply a default quantity to all the domain members. Policy and Quantity fields specify the same entries as the Quantity Policy and Quantity Value cells for individual domain members described above.

Multi-Sel

Type Boolean. Determines whether a selection point is single- or multi-select.

False indicates that only one selection can be made on the selection point. Be sure to choose *Def Choice* if desired (explanation follows). In this sense, the Quantity property duplicates the Multi-Sel property (explanation follows).

True indicates that multiple selections can be made on the selection point. When True is selected, the input row Defaults appears in place of the Def Choice property, and the Defaults Editor is made available for setting the default choices, their default quantities and policies.

SP Min/Max

Click the SP Min/Max Edit button to open a dialog to specify:

- Selection point Minimum Number:

0 indicates that the selection point is optional.

$1 - n$ indicates that the selection point is required and must have at least 1 or n selections.

- Selection point Maximum Number:

Unbounded indicates that there is no limit to the number of discrete domain members the user can choose (up to all members available). For efficient model maintenance, use this setting when the number of domain members is expected to vary over the life of the model or during run time.

$1 - n$ indicates a static limit; 0 has no meaning.

- Expression—the minimum and maximum are determined by an expression. Choose the expression from the drop down.
- SQL Query—the minimum and maximum are determined by query of the indicated database. Click the ellipsis button to open an edit window in which to create the query. Be sure to indicate the database to query.

Note. The Configurator system looks for the selection value in the first column of the first row of the data returned by the SQL query. Be sure to create a query that returns only one column and one row.

- Explanation—Message that you want to appear when the min/max limits are violated, advising the user when not enough choices are made, or too many. Explanations can be parameterized to indicate the objects involved.

See [Chapter 3, “Understanding Modeling,” Creating Parameterized Explanations, page 44.](#)

DM Min/Max

Minimum and maximum limits on the quantity of each domain member that may be selected.

- Domain member Minimum quantity: The least quantity that can be specified of a domain member when selected.
- Domain member Maximum quantity: *Unbounded* indicates that there is no limit to a domain member’s allowable quantity.
- Expression—If you want the minimum or maximum quantity to be determined at run time by an expression, select *Expression* and then select the desired expression (already defined) from the drop down list.
- SQL Query—the minimum and maximum are determined by query of the indicated database. Click the ellipsis button to open an edit window in which to create the query. Be sure to indicate the database to query. row.

Note. The Configurator system looks for the selection value in the first column of the first row of the data returned by the SQL query. Be sure to create a query that returns only one column and one.

- Explanation—Message that you want to appear when the min/max limits are violated, advising the user when too few or too many of a domain member are selected. Explanations can be parameterized to indicate the objects involved.

See [Chapter 3, “Understanding Modeling,” Creating Parameterized Explanations, page 44.](#)

Optional

Type Boolean. Default is *False* (required). If *False* is set, the user must make a selection on the control before the configuration can be verified.

Note. The same condition applies if Use Min/Max is *True* and Minimum Number is not 0.) If Optional is *True* and Multi Sel is *False*, the item *None* will be automatically added to the control at compile time.

Def Choice

Type String. Optional. The item in the control that is pre-selected. Can be a domain member name or *None*, which means “none of these domain members”. If Optional (described below) is *True*, the <None> option is computer-selected. If Optional is *False* (the control is required), *None* has no meaning. This property is not available if either Quantity or Multi Sel is *True*, as these properties indicate that quantities greater than 1 are to be specified.

See [Chapter 5, “Creating Objects for the Model,” Creating a “None” Domain Member, page 73.](#)

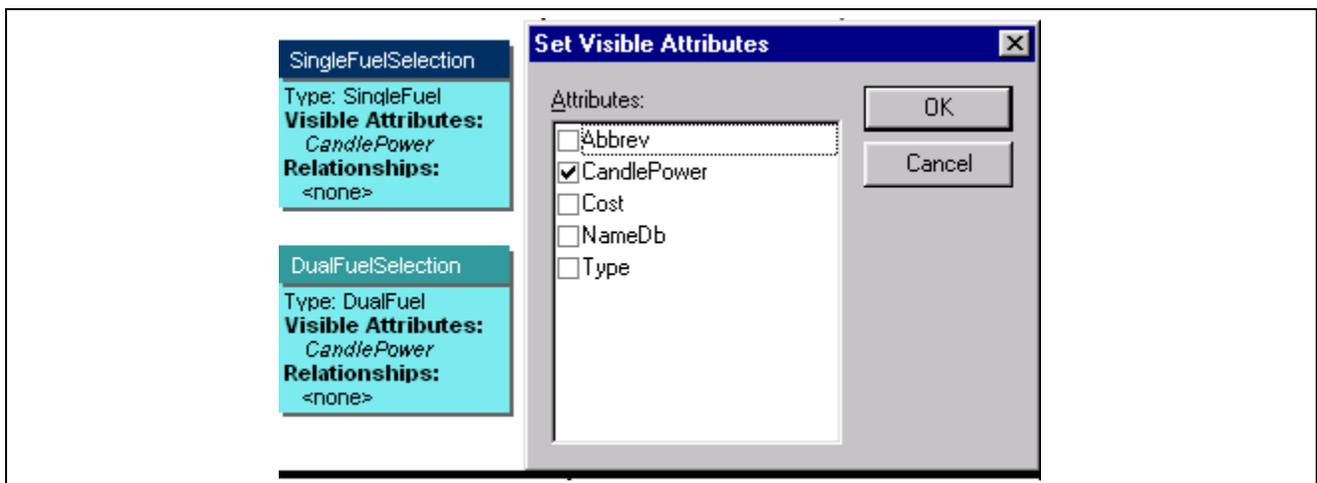
To view a model’s selection points, select View, Show, Selection Points or click the selection point icon in the toolbar. The selection points are displayed on the left side of the model structure view.

A selection point participates in all relationships that refer to the original parent class. If a relationship is made directly between a selection point and another object, that relationship is confined to the selection point. To add a selection point, drag a class from the components view into the selection point area, or drag a class from the Model Structure View into the selection point area.

To delete a selection point, select it in the Model Structure View, then press the Delete key. Once created, a selection point can participate in relationships with classes or class attributes members in the modeling editor.

Note the Properties Editor entries for selection points. Because selection points interface with the UI, settings made here determine how controls behave at run time. A selection point can be designated as multi-select (the user can select more than one of its items), and/or optional. If a control is optional, the user does not have to make a selection on the control. If a selection point is required, the model will not be considered valid until a selection is made. Selection point properties (Table Editor) New selection points appear at the bottom of the selection points area, and at the end of the selection points listing in the component view.

A class can have many attributes, but if attribute values are to be selectable at run time, they must be made visible from the selection point. To do this, right-click on a selection point that has attributes, select Set Visible Attributes, and check any attributes that you want to display at runtime. When an attribute is made visible, it will be shown on all the selection points for a class. The display of an attribute cannot be confined to certain controls.



Set visible attributes on selection points

Internalizing Data

External data can be extracted from the database and automatically inserted and defined in the model as for manually entered data.

This is a necessary step if the model is to be deployed in a mobile environment where configuration will take place locally on a standalone version of Advanced Configurator. The Configurator Administration Console has a more comprehensive internalization function than does Visual Modeler. It can internalize data stored in files other than those in a database.

Internalizing data is a “one time” operation. Once it is performed, changes to the data in the database will no longer be reflected in the model. To internalize data on an ongoing basis, use the option available with the compile command on the Server’s Administration page.

To internalize external data using the Visual Modeler, use the Project, Internalize Model command.

See Also

[Chapter 34, “Using Administration Tools,” Administration Console, page 415](#)

CHAPTER 6

Creating Relationships Between Model Objects

This chapter lists prerequisites and common elements and discusses how to:

- Create and edit expressions.
- Create externs.
- Create a relationship.
- Work with relationships.
- Create relationships outside the model with SQL queries.

Prerequisites

Visual Modeler provides graphical ways to express common configuration relationships. You can create relationships between classes, class attributes, selection points, and selection point attributes.

Before you create relationships, verify these setup requirements:

- To create a relationship by using internal domain members, the Internal flag must be set to *True* on the parent class.
- To create a relationship by using external data, the Internal flag must be set to *False* and you must have a valid database connection.

See [Chapter 4, “Setting Up the Modeling Environment,” Connecting to a Database from Visual Modeler, page 54](#).

- The Elimination and Comparison constraints and relationships operate on expressions; therefore, expressions of the proper type—numeric, Boolean, string, and date—must be defined before they can be created.

Common Elements Used in this Chapter



Click the Compatibility Constraint button to create a constraint that identifies all *valid* combinations and eliminates all other possibilities. Click the button, move the cursor to the first object in the desired relationship, and then click and drag to the second object. You can set up compatibility constraints between two classes and between a class and a selection point. However, you can't set up a relationship between a selection point and a class.



Click the Requirement Constraint button to create a constraint that causes a default quantity to be selected or a default selection to be made. Click the

button, move the cursor to the first object in the desired relationship, and then click and drag to the second object. You can set up requirement constraints between two classes and between a class and a selection point. However, you can't set up a relationship between a selection point and a class.



Click the Resource Constraint button to create a constraint based on the required quantities for particular selection. Click the button, move the cursor to the first object in the desired relationship, and then click and drag to the second object. You can define a resource constraint between two classes, two selection points, and between a class and a selection point. However, you can't set up a relationship between a selection point and a class.



Click the Summation Constraint button to create a constraint based on the sums of the values of class attributes. Click the button, move the cursor to the first object in the desired relationship, and then click and drag to the second object. You can set up summation constraints between two classes, two selection points, and between a class and a selection point. However, you can't set up a relationship between a selection point and a class.



Click the Dynamic Default Constraint button to create a constraint based on values determined at run time. Click the button, move the cursor to the first object in the desired relationship, and then click and drag to the second object. You can set up dynamic default constraints between two classes, two selection points, and between a class and a selection point. However, you can't set up a relationship between a selection point and a class.



Click the Expression button to open a dialog box to create a new expression.



Click the Elimination Relationship button to create a relationship that eliminates specified domain members when another domain member is selected.



Click the Comparison Relationship button to create a relationship that is executed when two values are compared. Output is either of numeric or Boolean type.



Click the Selection Points button in the Insertable Objects window of the expression editor, to insert a selection point object in the expression.



In the Insertable Objects window of the expression editor, signifies a domain member object that holds the value of the specified attribute from an extern.



In the Insertable Objects window of the expression editor, signifies an extern variable.



In the Insertable Objects window of the expression editor, signifies an expression that returns a date.



In the Insertable Objects window of the expression editor, signifies an expression that returns a numeric value.



In the Insertable Objects window of the expression editor, signifies an expression that returns a Boolean value.

Creating and Editing Expressions

An expression defines a variable. It associates a numeric, Boolean, date, or string value with a name. All relationships use expressions except summations and resource constraints.

An expression can contain:

- A literal value (to act as a constant).
- Selection point attribute values.
- A value that a user enters at runtime.
- Functions that operate on other expressions.

When you create an expression:

- The name appears in the Relationships or Expressions folder.
- The file name appears in the file list in the File view.
- The file name is automatically added to the project's .csp file.

There are a few points to remember while using the expression editor. When you create an expression, the numeric, Boolean, date, or string value within it is assigned to the expression name. Data for use in expressions can come from constants, class attributes, and other expressions.

Expressions have their own editor, which opens when you click the Expression editor button on the toolbar. The expression editor has these elements:

Return Type	Select the expression type to return values of types: <i>Numeric</i> , <i>Boolean</i> , <i>Date</i> , or <i>String</i> .
Expression	Displays the expression. Insert an attribute or expression name into this window, select a name in the Insertable Objects list, and drag it, or double-click it. Large or complex expressions can be formatted in a text editor and then pasted into this area. Edit carefully. There is no error checking on expressions until the model is compiled.
	<hr/> Note. You should stabilize selection point and attribute names before writing expressions. When an object name changes, Visual Modeler updates listings, selection points, and other places that the name is displayed; however, an existing expression can't be updated because it is text. An obsolete attribute or expression name will not be found at compile time. <hr/>
Name	Displays the expression name as defined.
Change Name	Opens a dialog box to change the expression name. Like other .cms objects, an expression is given a file name at the time of creation. This file name does not change when you edit the description name. The project manages the expression by the original file name.
Function Category	Select the general type of function to restrict the list of available functions in the Function Name list box to those that are appropriate for the expression type—Boolean, date, numeric, string, and user-defined. Or, click All to display the entire list of functions.

Function Name Select the function by scrolling and selecting, or by clicking in the list box and typing the name of the function. The list box auto-scrolls to the function as you type its name. To narrow the list of functions to those that are available to its data type, click a function type in the Function Category list box.

Insertable Objects Lists all selection points and their attributes. It also lists all expressions that have been created thus far.

Note. The selection point name is shown for reference. Expressions operate on attributes only. To insert an attribute or expression into the Expression window, double-click its name.

Find Enter text to search the Insertable Objects list from top to bottom. This field performs a character match as you type.

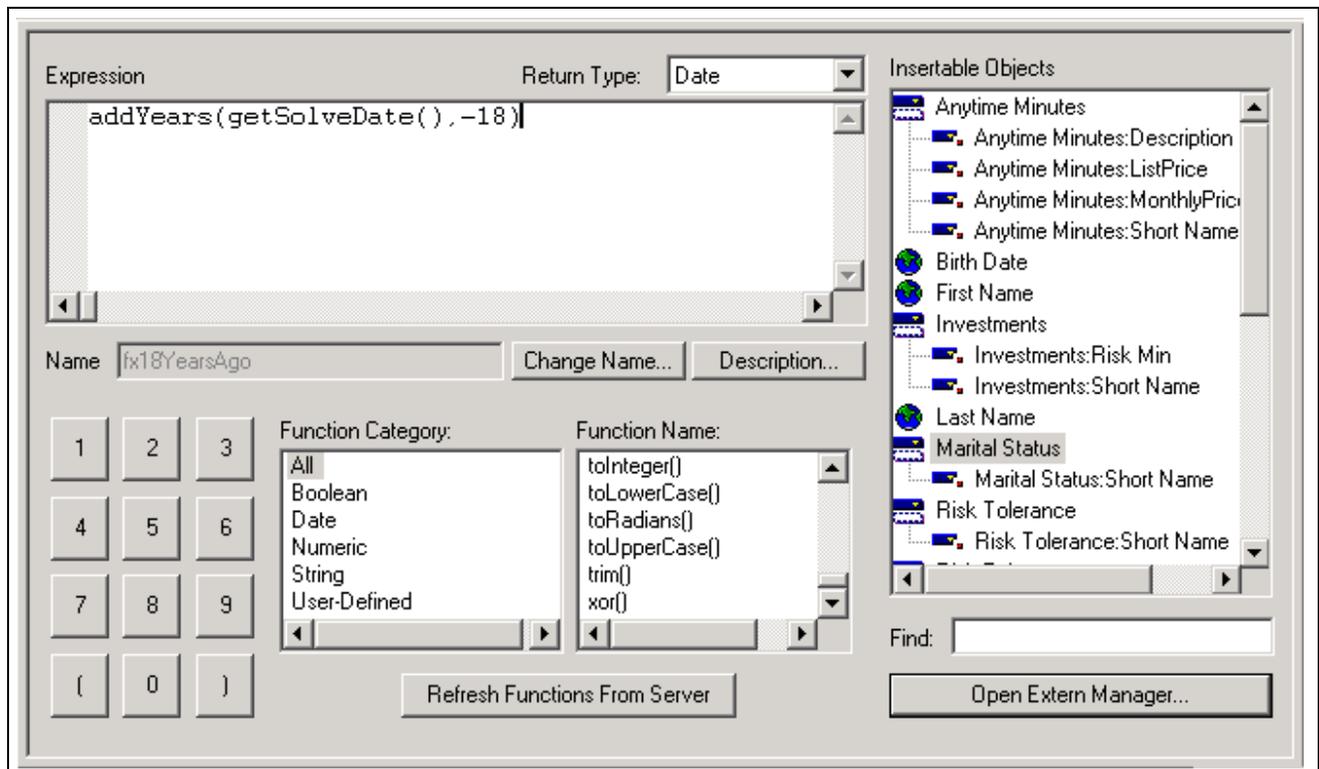
Note. This feature isn't active until the list is long enough to scroll.

Refresh Functions From Server Click to retrieve user-defined functions (if any were defined) from the Configurator server. The Configurator server that the system accesses is specified in the Project Settings dialog box.

See [Chapter 4, "Setting Up the Modeling Environment," Specifying Model Project Settings, page 60.](#)

Open Extern Manager Click to open the Extern Manager and add new extern variables to the Insertable Object list.

The following figure shows the expression editor window.



Expression editor

To create an expression:

1. Click the Expression editor button.
A name dialog box appears in which you name the new expression.
2. Click OK; the editor opens.
3. Write the expression in the Expression field by double-clicking the functions and insertable objects.
Enter additional information as necessary. Indicate the data type of the result in the Return Type drop-down list box.

Warning! Limit attribute values to 10 characters or fewer. Otherwise, the model will not compile.

See [Appendix A, “Visual Modeler Expression Editor Functions,” page 433](#).

To delete a relationship or expression, select its name in the Components view and press the DEL key, or right-click the name and select Delete.

Static Variables

A static variable is an expression that performs no operations and always returns a constant value.

To create a static variable, create an expression, set the return type, enter a value, and then close the window. (You cannot make an explicit assignment, such as *SalesTax=9.5*, inside the expression editor.) For example, to create an expression named *Airport Tax*, set the return type to *Numeric* and enter the value *12* in the Expression window.

External Variables

Use the external variable to retrieve a value—float, numeric, string, Boolean, or date—from the user or a database at runtime.

You cannot assign a value to a variable and operate on it in the same expression. Any calculated value must be assigned to its own expression name. Expressions can be built by using many other expressions.

User-Defined Functions

If you do not find a function that is appropriate to your specific needs, you can create one that is. To create your own function, you must create a description of the function in XML and write a Java class that implements the interface of Advanced Configurator. Sample code and sample files are provided.

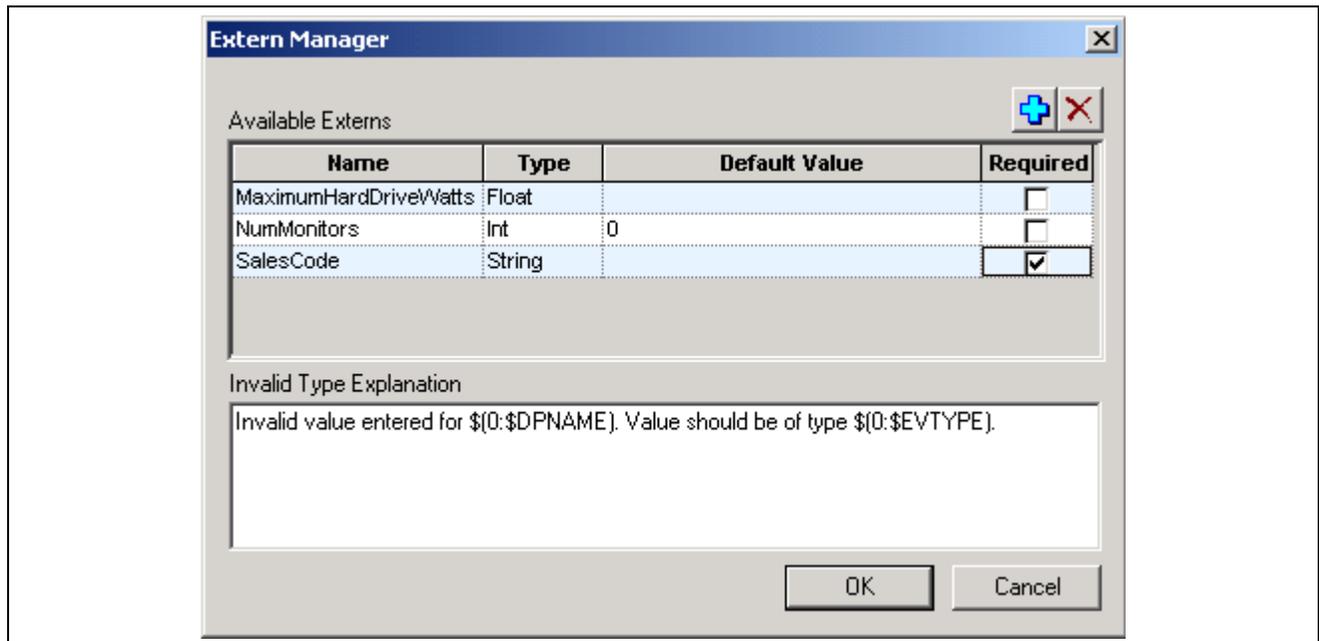
See [Appendix B, “Creating and Adding User-Defined Functions,” page 443](#).

To display the user-defined functions in the function list of the Expression editor, click the Refresh Functions from Server button and compile the model.

Creating Externs

You can direct the model to pull data at runtime from outside sources such as the enterprise database. Externs can have the following types: float, integer, date, string, and Boolean. The modeler can specify a default value for each extern and specify whether the passed-in value is required. Defaults can be overridden by end-user input on the user interface (UI).

You can declare externs in the Extern Manager dialog box. The Extern Manager lists all externally referenced data for the component model:



Extern Manager dialog box

Externs participate in the right-hand side of requirement constraints. Each extern has a Required check box in the cells of its column, which, when selected, makes the extern required if the left-hand side conditions are satisfied. The Required check box is the only setting that is available for the constraint-level use of externs.

Extern-based controls display violations and Control Why Help when the extern is declared as an argument or as part of an expression that serves as an argument. The arguments can be in a variety of relationships, including eliminations, comparisons, requirements, and compatibility constraints. Externs that are marked *required* at the object level in the Extern Manager display violations and Control Why Help only when running the completeness check.

At runtime, the system checks each type of extern for accuracy, but only date externs are also checked for formatting errors. If the value is not of the right type or format, an error message is displayed to the end user.

Because the requirement of an extern doesn't affect propagation, the requirement of externs is determined after static and dynamic default selections are applied.

To declare an extern:

1. Select Project, Extern Manager to open the Extern Manager dialog box.
2. Click the Add button to add a new row to the table.
3. Specify the properties of each:

Type Select *Boolean*, *Date*, *Float*, *Int*, or *String*.

Default Value Enter a default value to appear in the UI when the configuration session begins or when it is reset.

Required Select this check box if you want the extern to be required on a model-global basis. To make an extern required when constraint conditions are met, do not select the Required option in the Extern Manager, but rather in the constraint itself by inserting the extern as

a column in the right-hand side of the relationship and selecting the Required check box in the cells.

Invalid Type Explanation

Enter the message that you want to appear to the user when the type of the data does not match that specified for the extern variable. You can define parameters for explanations to indicate the objects involved.

See [Chapter 3, “Understanding Modeling,” Creating Parameterized Explanations, page 44.](#)

4. Click OK.

The externs now appear in lists of objects for building requirement constraints.

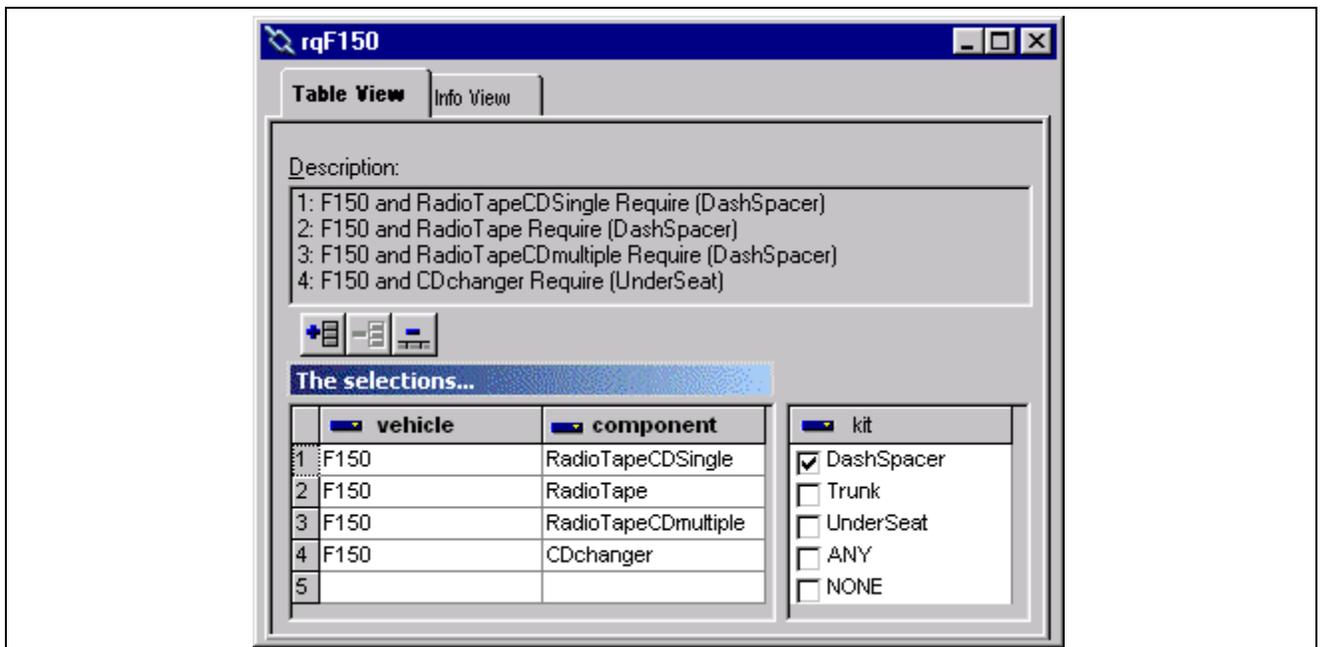
Creating a Relationship

You can start a relationship from either the Insert menu or the relationship palette of icons on the right margin of the application window. Depending on the type of relationship, one of two editors appears:

- The table-based editor, for compatibility and requirement constraints, and for dynamic defaults.
- The participant-list editor, for resource constraints and summation relationships.

The dialog boxes for compatibility constraints, requirement constraints, and dynamic defaults provide an opportunity to name the relationship and select the class or domain member attributes to constrain against. These relationships are created by using a table interface.

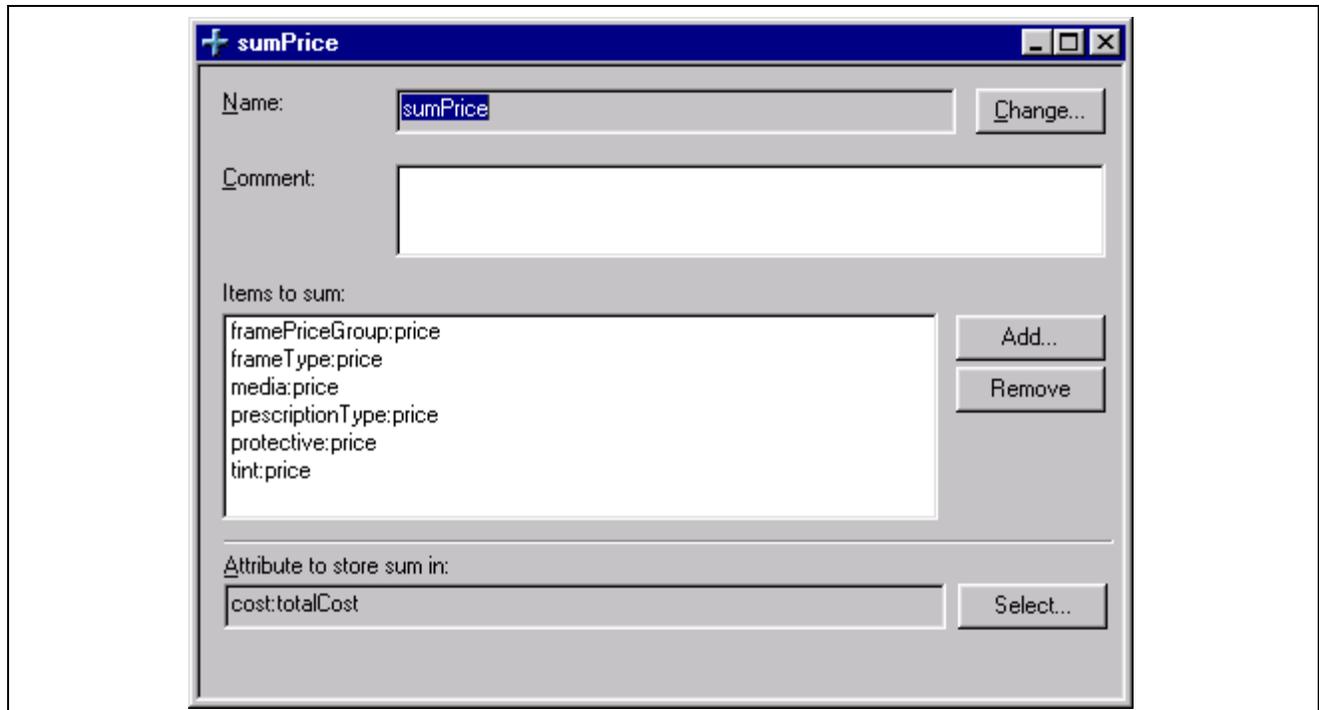
See [Chapter 3, “Understanding Modeling,” Relationships Between Objects, page 14.](#)



Requirement dialog box

The dialog boxes for resource constraint and summation relationships differ slightly because each participating object takes a specific role in the constraint. These constraints are created by using a simple list of participants.

See [Chapter 3, “Understanding Modeling,” Relationships Between Objects, page 14.](#)



The dialog box for a summation

To launch the relationship editor:

1. From the Insert menu, select a relationship.

Alternatively, click a relationship icon on the palette.

If the relationship is an elimination or a comparison, the relationship editor appears immediately.

For all other relationships, you can draw a line to indicate the relationship after the relationship icon is selected. Position the cursor over a class; the class is highlighted. Click and drag to draw a line to another class; when the target class is highlighted, release the button. You can also draw a line from a class to a selection point.

2. Provide a name for the relationship.

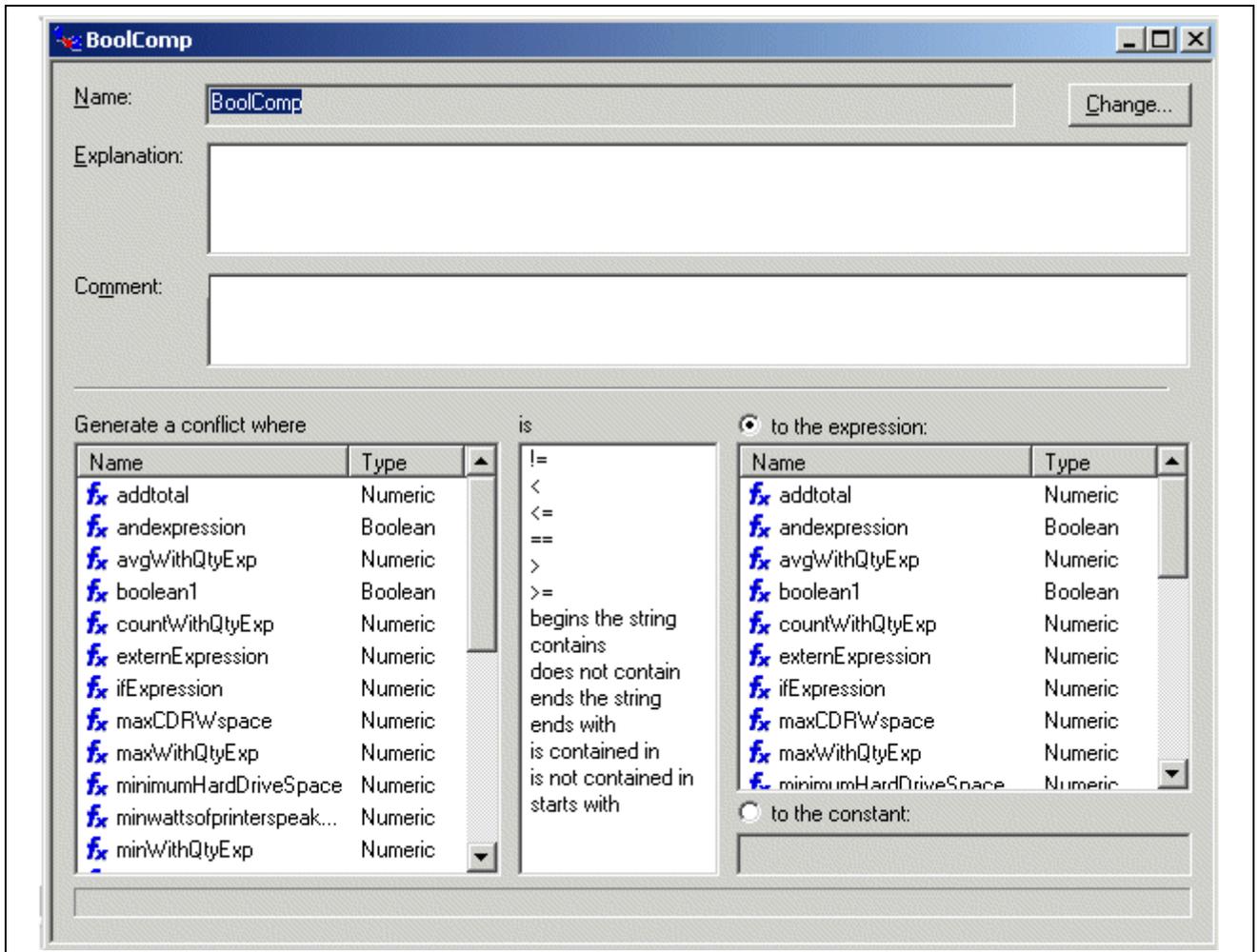
Note that the relationship name is assigned to a corresponding .cms file. To change a relationship name, click the Change button.

The dialog boxes for compatibility constraints, requirement constraints, and dynamic defaults provide an opportunity to name the relationship and select the class or domain member attributes to constrain against. These relationships are created by using a table similar to that shown in the example that follows.

See [Chapter 3, “Understanding Modeling,” Relationships Between Objects, page 14.](#)

Creating Relationships that Have Expressions

All relationships use expressions, with the exception of summations and resource relationships. The figures below show a numeric elimination, numeric comparison, and Boolean comparisons, respectively. To create a comparison or elimination relationship, click its icon on the palette. The comparison editor lists operators and available expressions; the elimination editor lists operators, selection points, and expressions. From these lists you can build an elimination or comparison expression that determines when a conflict will occur. If you have no expressions of a given type, the corresponding editor field is empty.



Comparison editor before expressions are selected

To include an expression in either the left- or right-hand side of the relationship:

1. Click the heading of the column to the right of where you want to insert the expression.
2. Click the Add Column button.

The Object Selection window displays, listing the available objects.
3. Scroll to the expressions, which are at the end of the list, and double-click the one that you want.

It is inserted into the new column.
4. If the relationship is a requirement constraint, dynamic default, or a compatibility constraint, define the relationship for each domain member:
 - a. Click in the cell of a domain member column to open a drop-down list box that contains the allowed domain member options.

For instance, rows of the selection point columns contain lists of domain members; expression columns contain subcolumns for operators, expression type (function or constant), and value.
 - b. In the same row as the domain member, click the cells of the expression subcolumns.

The first subcolumn, beginning left to right, contains a list of the allowable operators for the expression. The second subcolumn is the argument type, constant or function $f(x)$, of the value that is operated on. The third column is the value. If you chose the constant type in the preceding subcolumn, type the value. Be

sure that it is the correct data type for the expression. If you chose the function argument type, use the cell drop-down list box to insert the expression.

Note. The allowed choices for comparison operators depend on the return type of the expression argument. Similarly, allowed choices for the constant values depend on the expression argument's return type. Selecting an operator with the function symbol $f(x)$ narrows the allowed choices in the values subcolumn to a list of expressions that are compatible with the column's expression type.

- c. Repeat these steps for each domain member.
5. On the other side of the relationship, select the argument for the relationship from the list for the domain member or selection point.

Note. You can't add expressions to the right-hand side (RHS) of a compatibility, requirement, or dynamic default constraint.

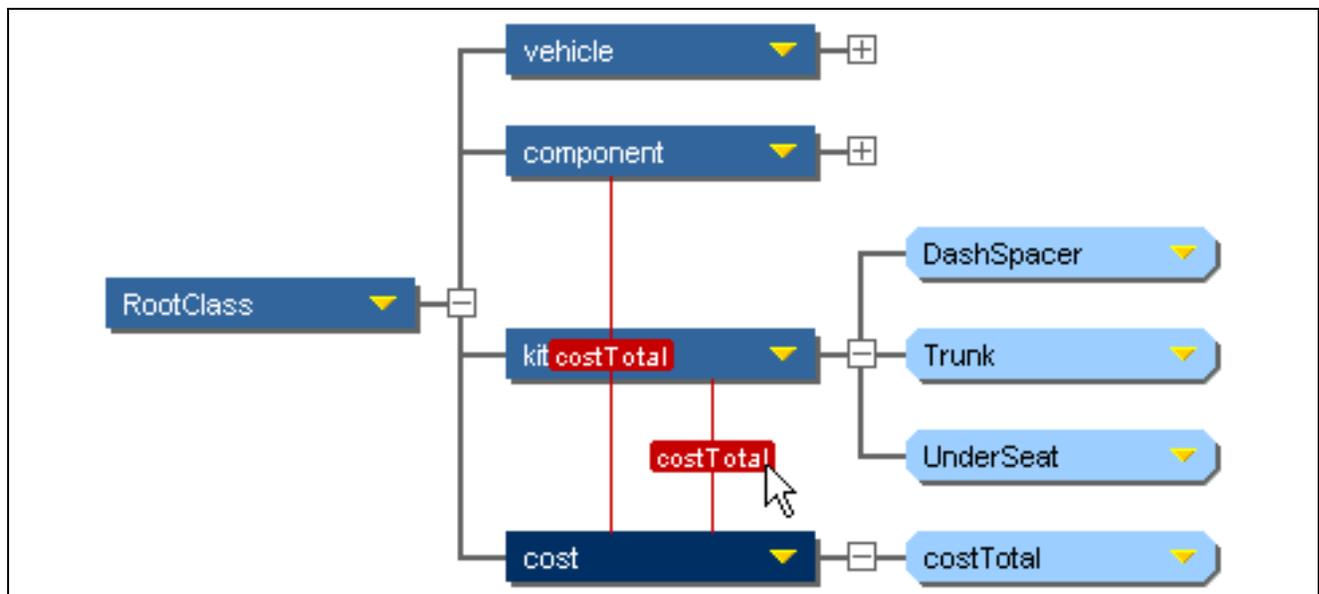
Working with Relationships

When a relationship is created:

- The name appears in the Relationships folder.
- The file name appears in the file list in the File view.
- The file name is automatically added to the project's .csp file.

To delete a relationship or expression, select its name in the Components view and press the DEL key, or right-click the name and select Delete. The constraint is removed from its .cms file. If the .cms file is then empty, you can delete it and manually remove the file name from the .csp file from the File view.

To display relationships in the Model Structure view, click a participating object, such as a class. Red lines appear, connecting all participants. The relationship name appears on the connecting lines. Double-clicking the name opens the relationship editor.



The costTotal relationship between subclasses cost and kit and subclasses cost and component

When an object is selected, the Attributes tab shows the relationship or expression name and the path to the file containing the relationship.

To reopen an editor, double-click its name in the Components tab.

Relationships Displayed as a Table

Compatibility, requirement, and dynamic default relationships are created by using a table-based editor.

The Relationship Editor toolbar appears directly above the table. The buttons are:



Add Column button. Add a column to the right of the current selection.



Add Row button. Add a row below the current selection.



Delete Column button. Click to delete the selected column.



Delete Row button. Click to delete the selected row.



Direction button. Toggle the directional bar in the current editor.



Toggle Compatibility button. Click to toggle the compatibility direction of the expression.



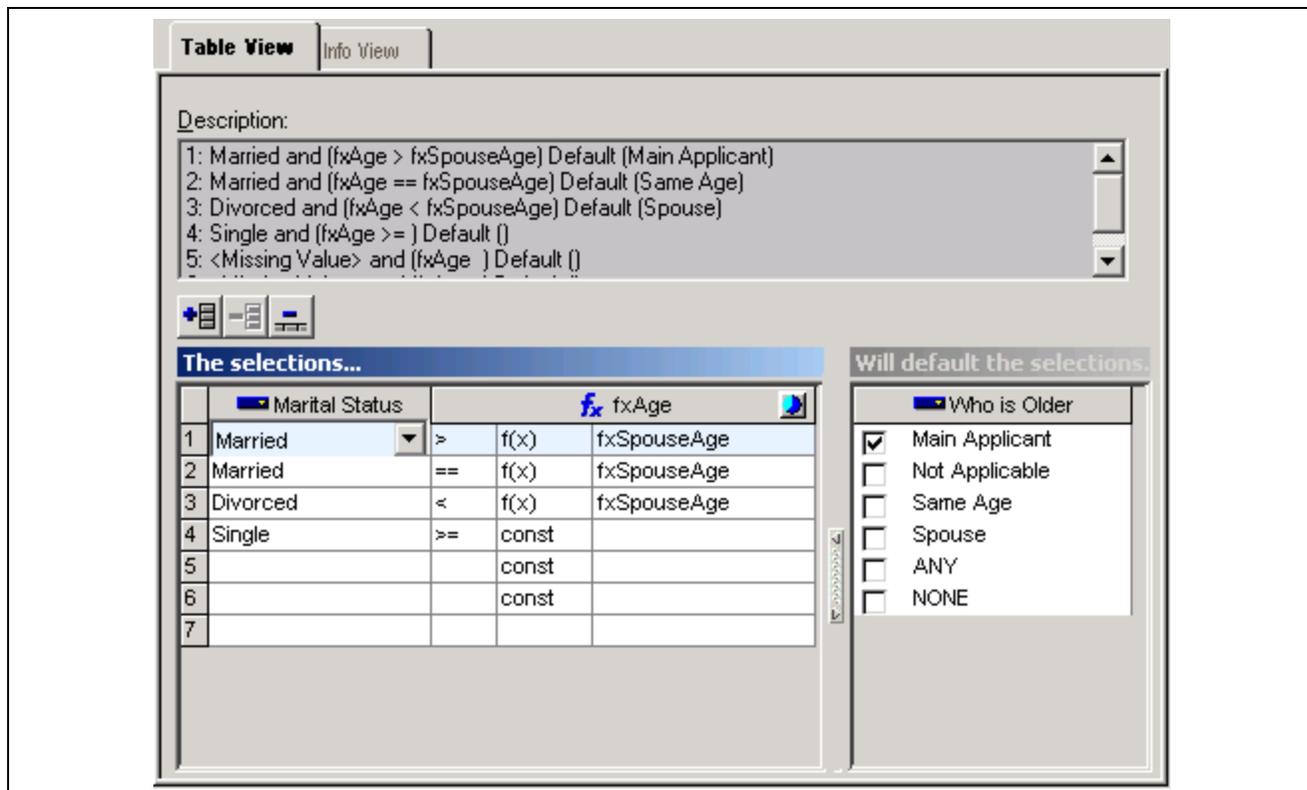
Clear LHS button. Clear all selected items on the left-hand side.

When inserting columns in a directional constraint, click to the right or left of the bar before clicking the Add Column button.

To resize columns, position the cursor over a header's vertical border and then drag left or right.

To move a column, position the cursor over its name, then drag the column to a new position; a red vertical line appears when a possible location is reached.

Note. The column of check boxes cannot be moved.



Dynamic default example using an expression on the left-hand side

In the preceding example of an expression, if the value of Married is greater than the value of SpouseAge, then the default selection is Main Applicant.

Table-based relationships have the Format property, which enables you to define them within a database table.

See [Chapter 6, “Creating Relationships Between Model Objects,” Creating Externs, page 91](#).

Expressions (Boolean, numeric, string, and date expressions) can participate as arguments (that is, columns) only on the left-hand side (LHS) of the expression; only the dynamic default can have expressions in both the left-hand and right-hand sides of table-based relationships. Each row of the expression column comprises two subcolumns—the comparison operator subcolumn followed by the values subcolumn. The allowed choices for the comparison operators depend on the return type of the expression argument. In turn, input validation for the constant values depends on the expression argument’s return type. To reference other expressions in the values subcolumn, the term $f(x)$ is attached to copies of the allowed comparison operators. If the relationship is on the RHS and you choose a $f(x)$ version of the operators, you narrow the allowed choices in the values subcolumn to a list of other expressions that are compatible with the column’s expression type.

Relationships Displayed with Participant Lists

For the resource constraint and summation relationship, the relationship editor displays a list of the attributes that are involved. They differ somewhat from table-based relationships in that each participating object takes a specific role in the constraint.

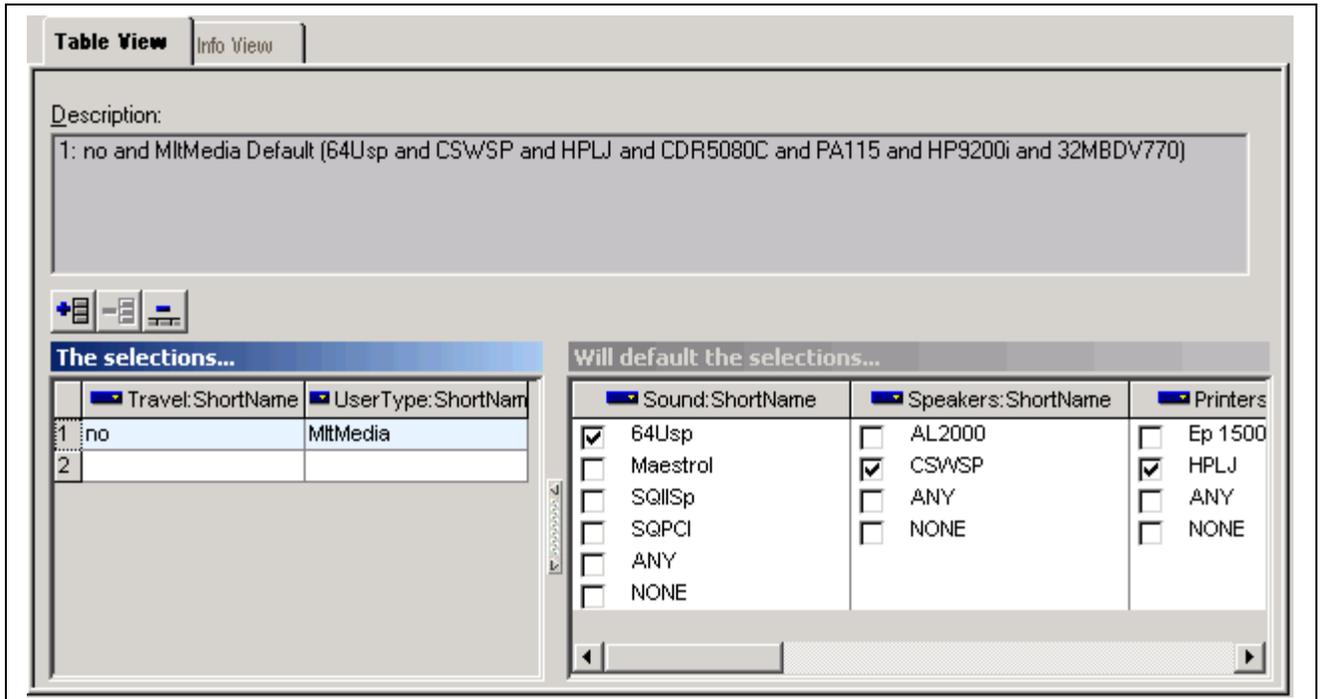
Editing Compatibility Constraints

In a compatibility constraint, the constraint type can be toggled from *Compatible* to *Not Compatible*. Typically, the majority of domain members are compatible. Check the Description window to verify that you have the expected setting.

ANY and ALL

The selection option ANY can appear in each row and column where internal data is used. ALL appears where external data is used.

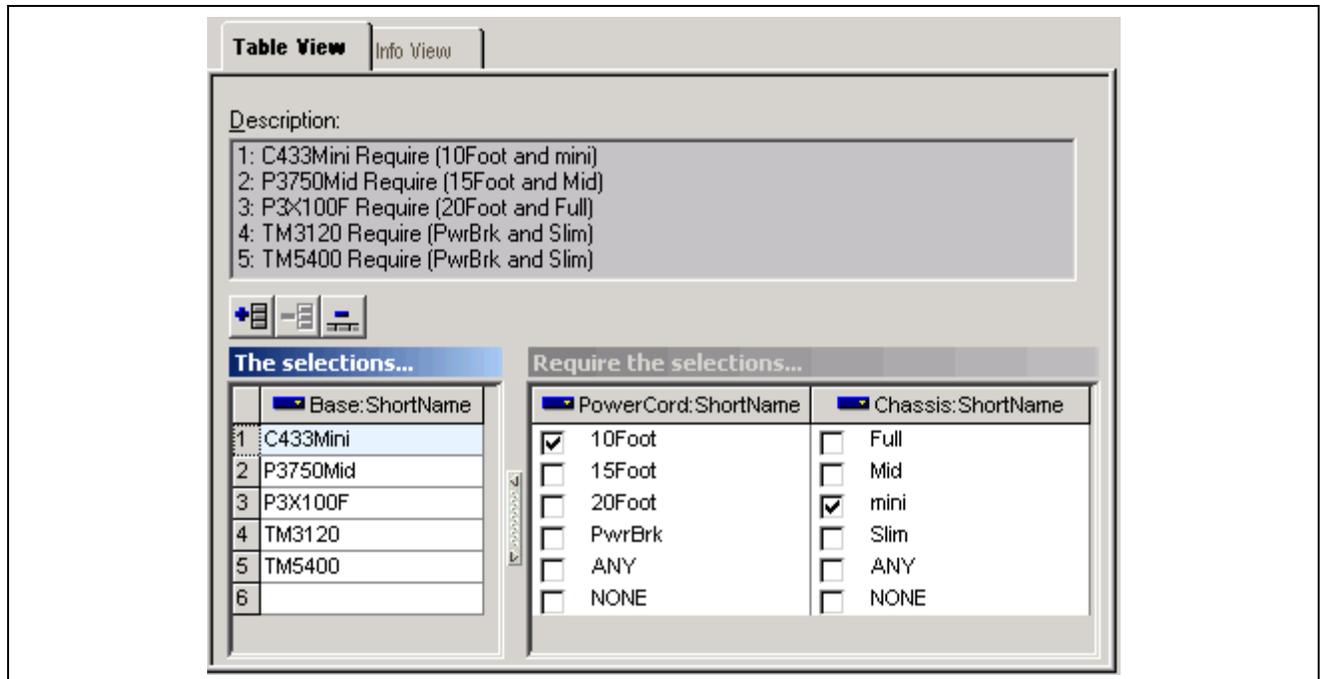
Note. ALL does not exist in the database; it is merely a shorthand representation that Visual Modeler displays so that you can select all queried domain members. ALL implies that any queried value is acceptable.



Compatibility constraint

Editing Requirement Constraints

The requirement constraint is displayed in a table similar to the compatibility constraint. ANY and ALL behave as they do in compatibility constraints.



Requirement constraint

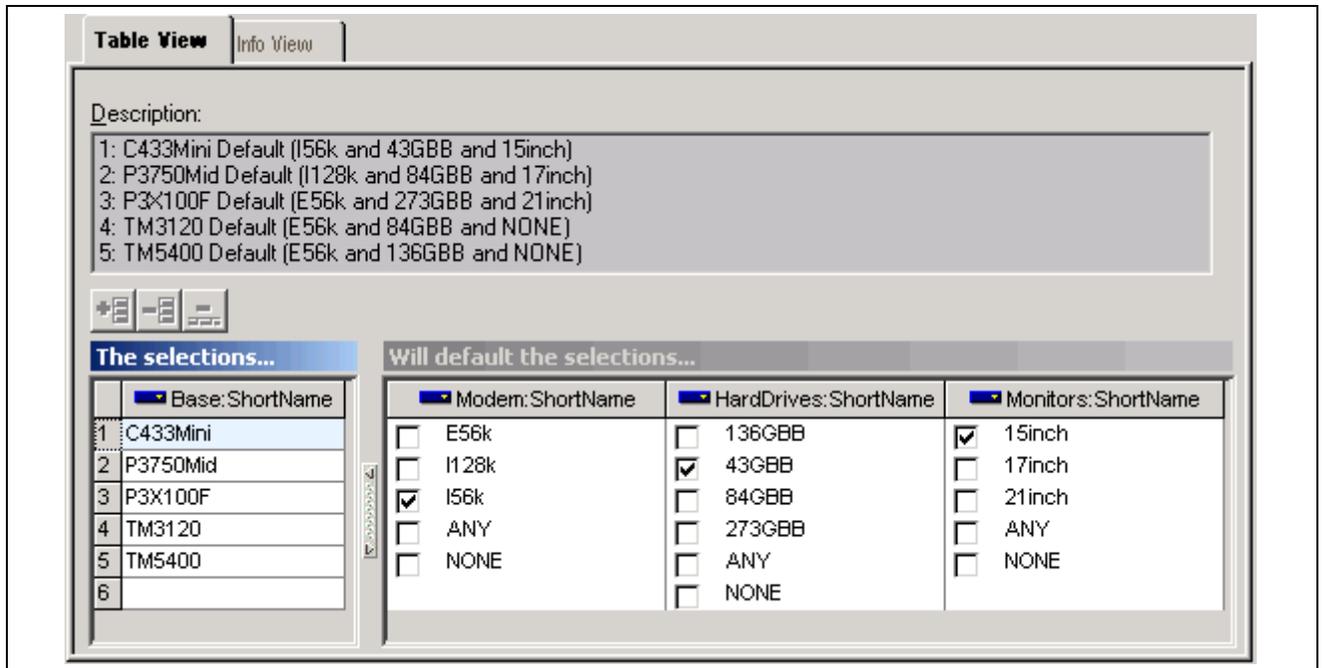
See Also

Chapter 3, “Understanding Modeling,” Requirement Constraint, page 18

Editing Dynamic Defaults

A dynamic default is a way to specify that a particular item will be preselected in response to a runtime event, such as a user selection. An end user can override a dynamic default without violating a constraint. (If a user overrides a constraint, a violation occurs, which can prevent a user from completing a valid configuration unless the selections are altered.) Regardless of the default, if all but one choice is constrained away, the remaining choice will be computer-selected. This behavior occurs automatically; no programming is required.

The dynamic default relationship table is similar to that of the compatibility or requirement constraints. ANY has the same significance. Dynamic defaults include the option NONE for every row. Selecting NONE means that there are no defaults for that combination of domain members, or that every combination of class and domain member properties is to be ignored. NONE allows you to work with queried values without knowing the specific domain members.



Dynamic Default

Note. Do not confuse the relationship editor NONE option with the None that is displayed at runtime.

See Chapter 5, “Creating Objects for the Model,” Creating a “None” Domain Member, page 73.

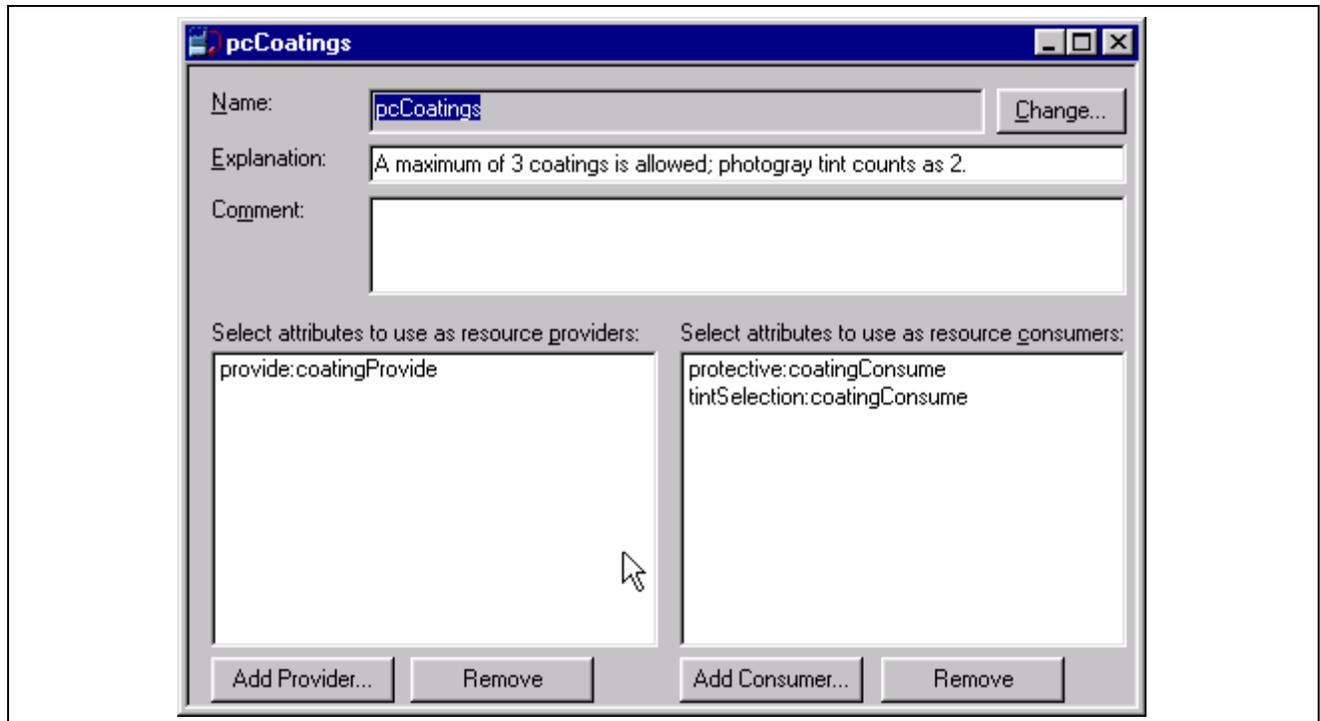
By default, the Comment area displays an equation for each selected combination.

Editing Resource Constraints

For the resource constraint, the relationship editor initially displays the classes or attributes that were selected or defined when the constraint was created. However, the system allows multiple providers and consumers. Clicking the Add Provider, Add Consumer, or Remove button displays a list of all attributes in the model.

Note. For the constraint to work correctly at runtime, the provider must be a selection point. The Model Tester automatically supplies controls for the provider and consumer so that you can monitor resources as choices are made.

The following figure shows an example in which a lens is arbitrarily assigned as a resource that can contain no more than three coatings. The coatings might take the form of a tint or other coating type, such as an anti-reflective coating.



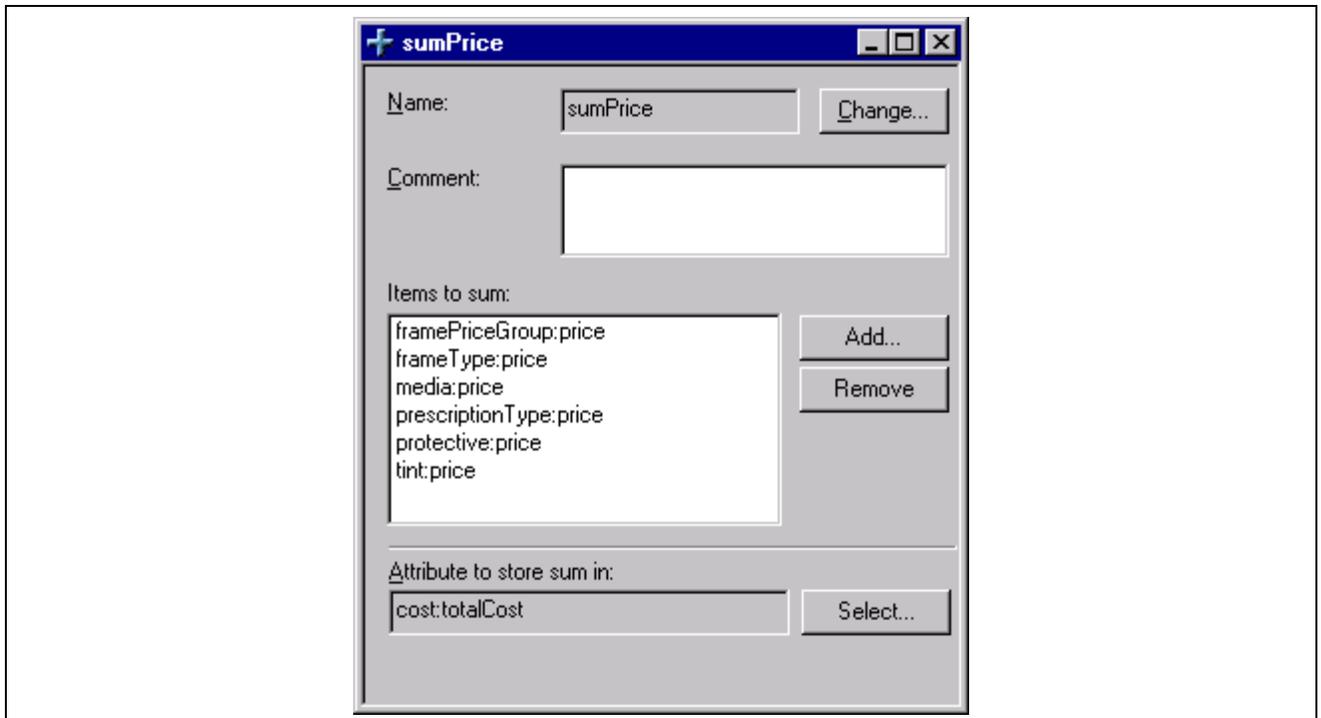
Lens resource and coating consumer

Editing Summation Relationships

Attribute summation uses a dialog box similar to that of the resource provider relationship. It simply lists all of the classes that contain the attribute that you want to sum. The total is placed in a domain member attribute on the class specified in the Attribute to store sum in field. The sum automatically appears in the Model Tester.

Note. The attribute that is used to store the sum must be different from the attribute that is totaled, or it must be a child of a class that is not participating in the summation relationship.

A summation constraint cannot calculate quantity. Instead, you must use an expression that uses `sumWithQty()`. An expression can use `sum()` or the `+` operator to accomplish the same task as this relationship. An expression has the additional capability to sum external variables.



Sum relationship for cost

Editing Elimination Constraints

An elimination compares a specific attribute value on a selection point with the value of a numeric, Boolean, string, or date expression. For example, you might create an elimination constraint so that domain members of the selection point `HardDriveSelection` are eliminated if the value of the selected hard drive's `Watts` attribute exceeds the value passed in by `externExpression`.

The layout of the options in the elimination editor allows you to build a constraint in logical order. The following table details each of the editor's options:

Explanation	Enter the message that the user sees when the conditions that are defined in the elimination are met.
Comment	Enter internal notations.
Allow	Specify which members of a selection point are eliminated. Values in the Allow list vary depending on the type of attribute that is specified. <ul style="list-style-type: none"> <i>All Of:</i> The domain members that did not satisfy the condition are eliminated. Supported by all attribute types. <i>None Of:</i> The domain members that did satisfy the condition are eliminated. Supported by all attribute types. <i>The First Of:</i> All domain members are eliminated except the first domain member that satisfies the condition. Supported by Boolean and string types. <i>The Last Of:</i> All domain members are eliminated except the last domain member that satisfies the condition. Supported by Boolean and string types. <i>The Least Of:</i> All domain members are eliminated except the domain members that contain the smallest numeric value out of all domain members that satisfy the condition. For example, if A1, A3, A7, and A9 satisfy the condition with

attribute values of 3, 7, 3, and 9, then every domain member except A1 and A7 is eliminated. Supported by the numeric type only.

The Greatest Of: All domain members are eliminated except the domain members that contain the largest numeric value out of all domain members that satisfy the condition. For example, if A1, A3, A7, and A9 satisfy the condition with attribute values of 3, 7, 3, and 9, then every domain member except A9 is eliminated. Supported by the numeric type only.

The Earliest Of: Analogous to *The Least of*. Supported by the date type only.

The Latest Of: Analogous to *The Greatest of*. Supported by the date type only.

the members of	Select model selection points.
where	Select from a list of all domain members for the selection point that is selected in <i>the members of</i> column.
is	Select an operator. The list of available values varies with the domain member type.
to the expression	Select the expression option to specify the constraint's determining value at runtime.
to the constant	Select the constant option to set a known, static value. Use this option to test the constraint.

To create a definition, select:

1. A selection point from the members of column.
2. An attribute from the where column.
3. A comparison operator from the is column.
4. Either the to the expression or to the constant option.
5. A selection from the Allow column.

Repeat selections 2 through 4 for each attribute of the selection point class.

Note. The attribute type must match the expression or constant type.

During runtime, the system uses the comparison operator in the is column to compare the domain attribute value to the value that is supplied by the specified expression or by the constant.

If the comparison evaluates to true, the system adds the domain member to a list of domain members that satisfy the condition of the elimination.

Editing Comparison Constraints

The comparison editor, like the elimination editor, is designed so that you can build and view the constraint in logical sequence.

Like the elimination editor, the comparison relationship editor allows you to enter an arbitrary constant for comparison. You can use this constant instead of creating an expression to represent the value.

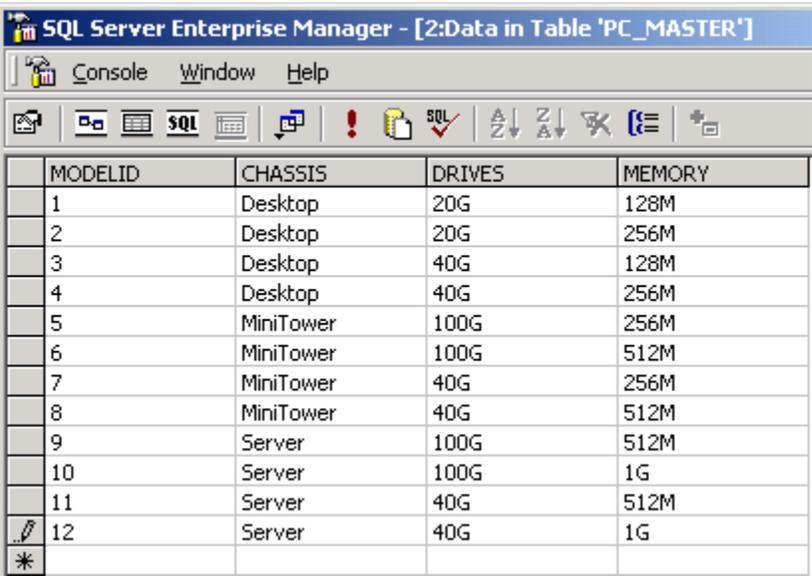
To create a comparison, select one item from each of the columns. Operators presented in the is column depend on the type of expression that is selected in the Generate a conflict where column.

Note. Attribute types for the expressions (and constant if used) must match.

Creating Relationships Outside the Model with SQL Queries

You can use the SQL query feature to define constraints. This feature removes the constraint definition from the model altogether and places it in a database.

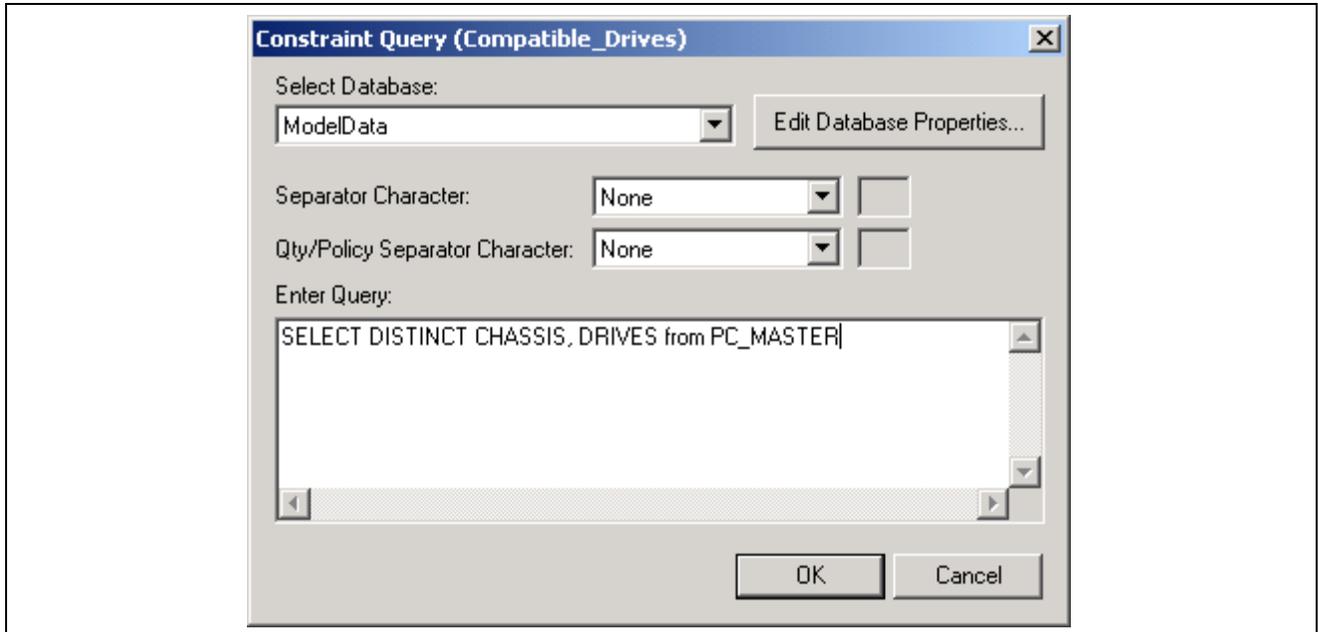
The SQL query uses a SELECT DISTINCT statement to dynamically retrieve the domain member combinations that are compatible. To update compatibilities, you edit the database entries rather than the constraint definition in the model, eliminating the need to recompile the model. Thus, SQL query constraints allow you to store the compatibility of product options in a SQL table and drive product logic dynamically from within a database.



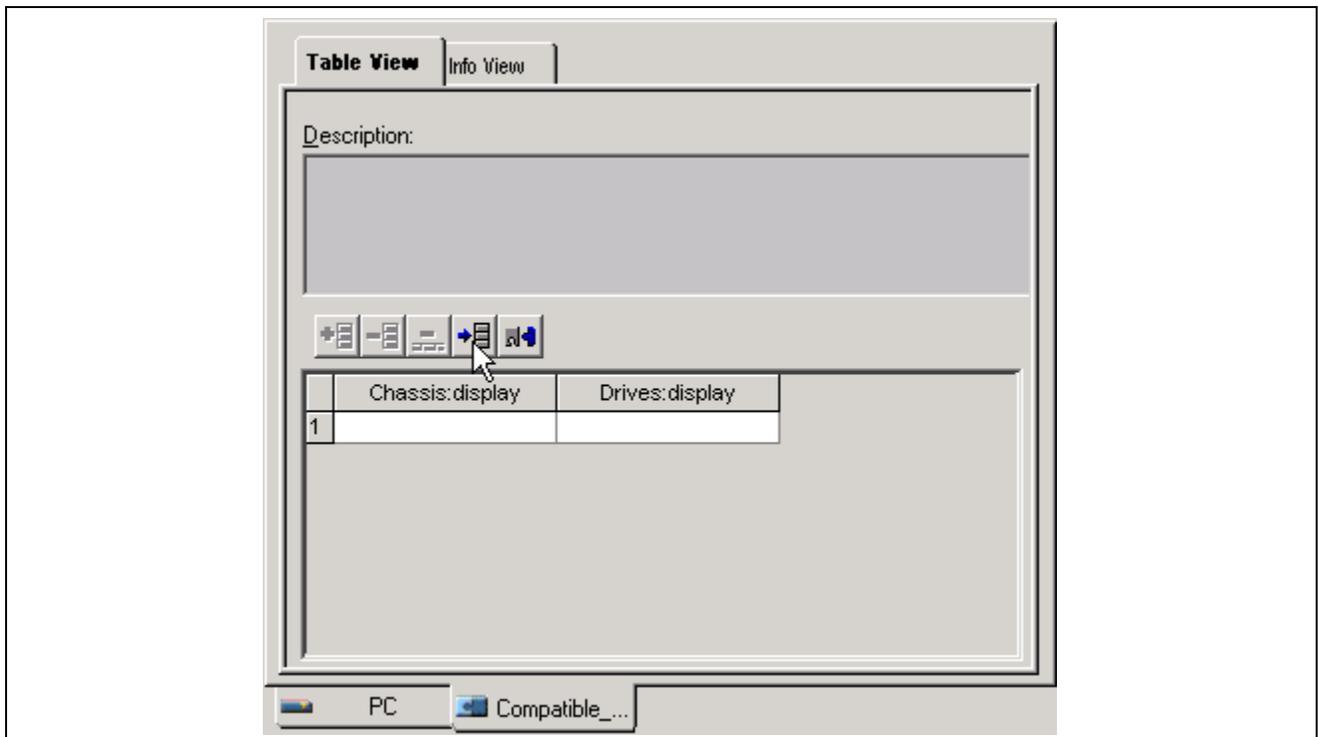
MODELID	CHASSIS	DRIVES	MEMORY
1	Desktop	20G	128M
2	Desktop	20G	256M
3	Desktop	40G	128M
4	Desktop	40G	256M
5	MiniTower	100G	256M
6	MiniTower	100G	512M
7	MiniTower	40G	256M
8	MiniTower	40G	512M
9	Server	100G	512M
10	Server	100G	1G
11	Server	40G	512M
12	Server	40G	1G

SQL query table containing all possible, valid configurations

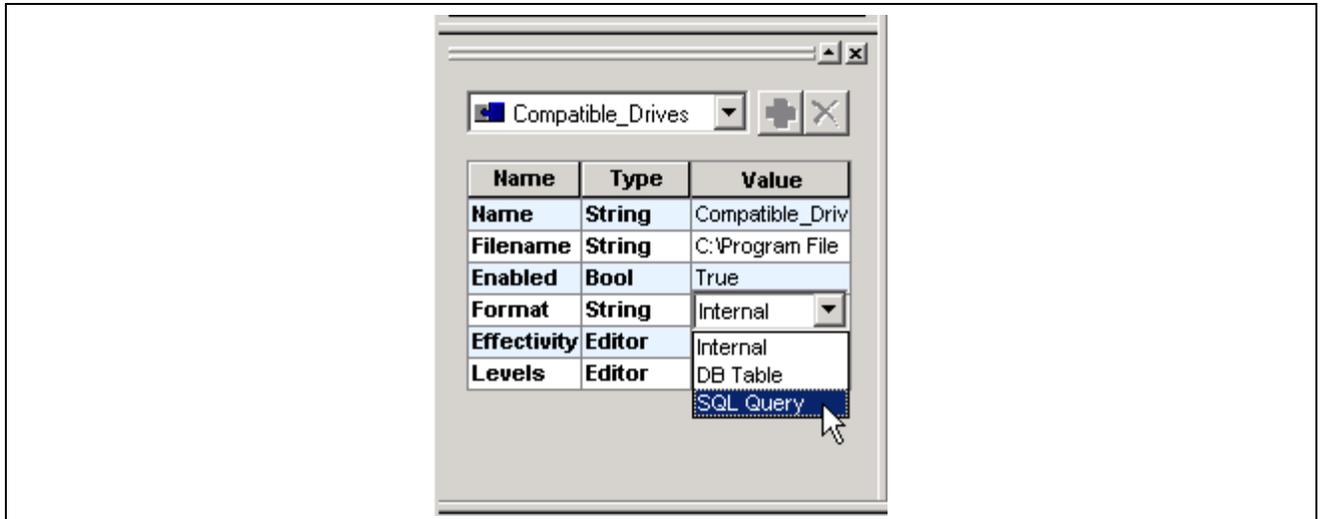
The first two rows of the example in the figure state: “Desktop comes with a 20G drive and in 128M and 256M Memory versions.”



Constraint Query dialog box showing a sample query



Changing the directionality of the constraint



Assigning the SQL Query property in the relationship properties panel

To create a SQL query constraint:

1. Set up the classes by associating each column in the table with an attribute in the class.

Note. Don't use the system-defined attribute Name as a constraint item. You must add your own attribute. In the example, an attribute was created called *chassis*, and then the value in the attribute that corresponds to the distinct values in the table was added. For example, the Desktop Chassis domain member has an attribute called *chassis*, and its value is Desktop. (Optional step) Set up a SQL Query for the domain members as well. The SQL statement for the class retrieves its data from the Chassis column in the table. Then, using a DISTINCT clause in the SQL statement, it retrieves all of the possible values for the domain member list. This step saves having to add these values manually.

2. Define the constraint:

- a. Create the constraint by using the compatibility constraint editor.

The New Compatibility Constraint dialog box appears first, in which you select the attributes.

Warning! In the New Compatibility Constraint dialog box, be sure to select the correct attribute for the constraint arguments, as this is critical for the constraint to work properly. The constraint will not work if you use the default constraint argument.

- b. Make the constraint nondirectional by clicking the button on the editor toolbar.
- c. Change the format in the Properties panel to *SQL Query*.

The Constraint Query dialog box appears.

Note. Table and column names are case-sensitive.

- d. Specify a standard SQL query for each column in the constraint, separating the queries with a comma. Make sure that the RHS columns and LHS columns of the constraint are present on the attribute that it is being mapped to, rather than just the selection point name. You can specify that a space or another separator be used in the query output to allow for current data formatting.
- e. Click OK. The constraint is populated with values.

If a dynamic default contains default quantities, you can specify the quantities in the database if the column refers to a selection point with quantity. Do so by separating the policy, quantity, and value using a user-definable quantity and policy separator. For example, if @ is the separator, an entry in the database would be MIN@4@Tires. The policy can be MIN OF, MAX OF, or SUM, and the quantity must be a number.

Note. The quantity cannot be an expression reference.

CHAPTER 7

Specifying Quantities on Selection Points

This chapter provides an overview of quantity setup and discusses how to:

- Specify the number of allowed selections and optional or required status.
- Specify single- or multi-select controls.
- Set quantity limits on domain members.
- Set default quantities and selections.
- Define a dynamic default quantity for a selection.
- Attach metadata to selection points.

Understanding Quantity Setup

After you have set up the model's classes and objects and defined their relationships, you are ready to define the properties on their selection points, including the model's quantities. Use the selection point's property table to define its quantities.

To view a model's selection points, select View, Show, Selection Points or click the View Selection Point button in the toolbar. Click the selection point to update the property table with its current property values.

Alternatively, you can click the selection point's name on the Components tab.

Setting up quantities involves these general steps:

Step	Reference
Specify whether a selection must be made from the selection point, and how many selections can or must be made.	See Chapter 7, "Specifying Quantities on Selection Points," Specifying the Number of Allowed Selections and Optional or Required Status, page 110.
Decide how many selections the user can make.	See Chapter 7, "Specifying Quantities on Selection Points," Specifying Single- or Multi-Select Controls, page 111.
(Optional) Set ranges for allowable domain member quantities.	See Chapter 7, "Specifying Quantities on Selection Points," Setting Quantity Limits on Domain Members, page 112.

Step	Reference
If you want one or more items to be preselected, specify which domain members you want to appear and in what quantity.	See Chapter 7, “Specifying Quantities on Selection Points,” Setting Default Selections and Quantities, page 114.
Define quantity application behavior for dynamic defaults and requirement constraints.	See Chapter 7, “Specifying Quantities on Selection Points,” Defining a Dynamic Default Quantity for a Selection, page 117.

Specifying the Number of Allowed Selections and Optional or Required Status

This section describes how to specify how many different items (domain members) within the selection point—not the quantity of each selected item—that the end user can make in the control that is associated with the selection point. In the process, you also specify its optional or required status.

For example, in a selection point that contains items A, B, C, and D, a minimum of 2 means that the end-user *must* choose at least two items, such as B and C or A, C, and D. Note that a minimum greater than 1 designates the selection point as multi-select. If minimum = 0, the end user isn’t required to choose any of the items—the selection point is considered optional.

Note. This section also describes how to specify single- and multi-select controls, and whether the end user is required to select at least one of its items. As described elsewhere in this chapter, you can use the Multi Sel and Optional properties to accomplish the same purpose. However, the Multi Sel and Optional properties do not allow you to specify a quantity; using the min/max quantity properties as described here gives you the full scope of the default quantity and min/max functionalities.

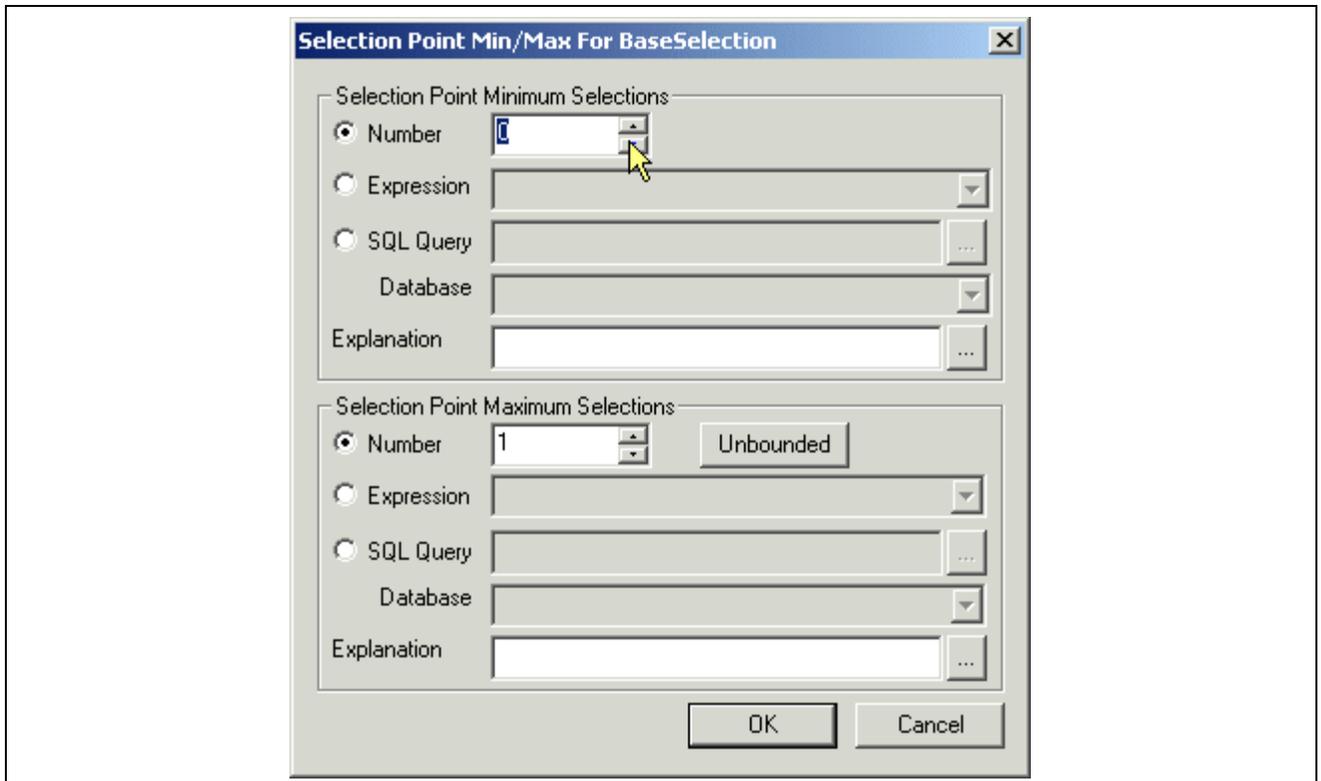
To specify the number of allowed selections, in the selection point’s property table, select *Use Min/Max = True*.

Name	Type	Value
Name	String	Size_1
Filename	String	c:\Program Files
Type	String	Size
Quantity	Bool	False
Use Min/	Bool	True 
SP Min/M	Editor	Edit...
Defaults	Editor	Edit...

Properties table for a selection point

Note. If you are not interested in defining minimum and maximum limits on the selection point, you can use the Optional property to specify the selection point type. (Optional = False specifies a required selection; Optional = True specifies an optional selection.)

Click the Edit button for SP Min/Max. The Selection Point Min/Max dialog box for the selection point appears:



Selection Point Min/Max dialog box for the selection point

Specify a minimum number of selections in one of three ways:

- *Static*: Enter in the Number list box either *0* (the user is not required to select from this selection point), or a quantity of *1* or greater, not to exceed the number of different items in the selection point.
- *Runtime, by expression*: If you want the number of selections to be determined at runtime by an expression, select *Expression* and select the expression that was created earlier from the drop-down list box.
- *Runtime, by SQL Query*: If you want the number of selections to be determined by stored data, select *SQL Query*, write the query in the entry box, and specify the database by name.

Note. Advanced Configurator looks for the selection value in question in the first column of the first row of the data returned by the SQL query. Thus, you should create a query that returns only one column and one row.

In the Explanation field, enter a message to be displayed to the user when the minimum and maximum quantities are not satisfied during runtime.

See [Chapter 3, “Understanding Modeling.” Creating Parameterized Explanations, page 44.](#)

Specifying Single- or Multi-Select Controls

You can designate a selection point’s single- or multi-select property in two ways:

- Set a minimum of 2 or more, as in the previous section.

You may, in fact, have already specified it.

- Set a maximum of 2 or more, as described in this section.

Note. If you do not intend to define quantities on the selection point, you can use the Multi Sel property to specify selection point type. (Multi Sel = False specifies a single-select selection point; Multi Sel = True specifies a multi-select selection point.)

When Use Min/Max is changed from False to True, the selection point’s properties provide the default values for the SP Min/Max and DM Min/Max dialog boxes. For example, if the selection point is Multi Sel=True, Optional=False, and Quantity=True, then changing Use Min/Max from False to True gives settings of SP Min = 1, SP Max = Unbounded, DM Min = 0, and DM Max = Unbounded.

To specify single- or multi-select, as you did for the selection point minimum, specify values for the maximum limit:

- Single-select: Number = 1.
- Multi-select: Number = 2 or greater, or *Unbounded*.

For multi-select controls, *Unbounded* allows the number of selections to be any number of domain members in the selection point. Using *Unbounded* makes model maintenance easier in cases where all domain members can be selected and the number of domain members may change over time.

In the Explanation field, write a message to be displayed to the user when the maximum number of selections is exceeded during runtime.

See [Chapter 3, “Understanding Modeling.” Creating Parameterized Explanations, page 44.](#)

Setting Quantity Limits on Domain Members

Use domain member limits to place a minimum or a maximum limit or both on the number of the selected domain member. These limits apply to all domain members in the selection point. You can set quantity limits on the domain members of both single- and multi-select selection points. In the following example, the end user can choose no more than three of any one item (it is a single-select control):

SingleSelectGroup.jsp template generated this form control

SingleSelectGroupElimLevel.jsp template generated this form control

Address: http://advicor.middle.ca:icc.com:7001/sample/CalcProcDefCfm.jsp

Model Name	Model ID	Compile Version
sample	1-0-0	20000229-100417-909

Next (page *)

[SampleConfig].BaseSelection

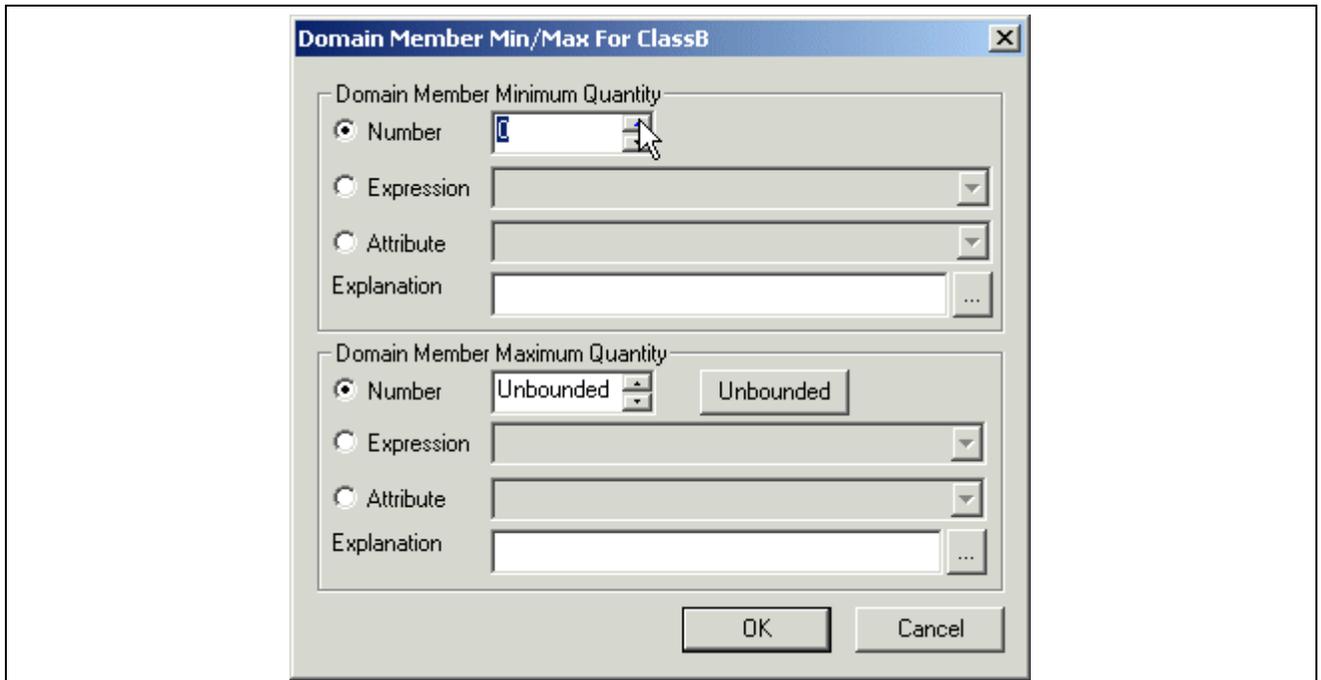
- Prima Base Celeron 433Mhz MiniTower
- Suprema Base Pentium III 750 Mhz MidTower
- Magna Base Pentium III Xeon 100 Mhz FullTower
- Advanta Base TM3120 400 Mhz Mobile PC
- Extra Base Tm5400 700 Mhz Mobile PC

[SampleConfig].BaseSelection

- Prima Base Celeron 433Mhz MiniTower

Example of single-select control

The following example shows the domain member for the Min/Max dialog box for the selection point ClassB:



Min/Max dialog box

Number	Enter a numeral to be used as an absolute number.
Expression	If you want the minimum quantity to be determined at runtime by an expression, select this option and then select the expression from the drop-down list box.
Attribute	If you want the minimum number to be determined by an attribute value, select this option and select the attribute from the drop-down list box.
Explanation	Enter a message to be displayed to the user when the minimum and maximum quantities are not satisfied during runtime. See Chapter 3, “Understanding Modeling,” Creating Parameterized Explanations, page 44.
Unbounded	Click this button to Indicate that there is no upper limit.

To set minimum and maximum limits on domain members:

1. On the selection point’s property table, set Quantity to *True*.
2. Set Use Min/Max to *True*.
3. Click the Edit button for DM Min/Max to access the Domain Member for Min/Max dialog box.
4. Select either *Number*, *Expression*, or *Attribute* to indicate the source of the value for the minimum quantity.
5. In the Explanation field, enter a message to be displayed to the user when the minimum and maximum quantities are not satisfied during runtime.

An example message is: “Please enter a quantity of at least 3 racks.” Be sure that the message is specific enough to be helpful but that it makes sense for all domain members in the selection point.

6. Enter settings for the maximum quantity as you did for the minimum quantity.
Use the Unbounded button to indicate that there is no upper limit on the quantity.

Setting Default Selections and Quantities

You can set the selection point to display one or more default selections with quantities, to be displayed when the end user starts the configuration. The source for these quantities can be:

- Explicit: The modeler specifies the default value in the model.
- Attribute: The modeler specifies an attribute of the domain member from which to retrieve the value at runtime.

Quantity Policy	Quantity Value/Expression	Domain Member
Overridable	0	64 Voice w/ Speakers
Overridable	0	Maestro Integrated Sound
Overridable	0	Super Quad II w/ Speakers
Overridable	0	Super Quad PC1 Card

Edit Default Choices dialog box

Quantity Policy	<p>Specify the quantity-choosing behavior to apply when constraints derive quantity values of their own in response to runtime input.</p> <p><i>Overridable, Overridable f(x)</i>: The quantity applied by the dynamic default or requirement constraint will be the final configuration quantity for that domain member.</p> <p>See Chapter 3, “Understanding Modeling.” Interaction between Default Quantities and Min/Max Settings at Run Time, page 42.</p> <p><i>Min of, Min of f(x)</i>: Apply the quantity value or expression result as a minimum quantity that will be selected as a default for the domain member.</p> <p><i>Max of, Max of f(x)</i>: Apply the quantity value or expression result as the upper limit of the quantity that will be applied to the domain member.</p> <p><i>Sum, Sum f(x)</i>: Add the quantity of each default selection of the domain member with the dynamic defaults and requirement constraints.</p> <p>For instance, if ItemA has a default of 2 and a dynamic default that specifies the quantity 2, then the runtime quantity of the domain member is 4.</p>
Quantity Value/Expression	Click in the cell of the desired domain member and select or enter a numeral (0 to n), or an expression that depends on the quantity policy that is selected. If you select an expression, the value resulting from the expression is used. 0 indicates that the domain member is not selected by default.
Domain Member	This field displays the name of the domain member in the selection point. This field is display-only.
Use Quantity Policy for All Domain Members	<p>Specifies whether and how to apply a default quantity to all the domain members.</p> <p>Policy and Quantity fields specify the same entries as the Quantity Policy and Quantity Value cells for individual domain members described above.</p>

Setting Explicit Default Choices and Quantities

To set up default selections and their quantities:

1. On the selection point’s property table, set Quantity to *True*.
The table adds the property Use Min/Max.
2. Click the Edit button for Defaults to display the Defaults editor.
3. Enter the desired quantity (1 to n) in the appropriate Quantity Value/Expression cell for the domain member.
0 indicates that the domain member is not selected by default.
4. In the selection point’s property table, set Use Min/Max to *True*.
5. Click the Defaults Edit button to open the Defaults editor dialog box again.
6. If you want the default quantity to be determined by an expression, select the f(x) version of the displayed quantity policy in that cell’s drop-down list box.
Return to the Quantity Value/Expression column and select the expression from the drop-down list box. (Expressions are defined in the Expression editor).

7. If there are dynamic default constraints or requirement constraints that could select a domain member during runtime, consider whether you want the default quantity entered in the table cell to remain in effect if one (or more) of the dynamic defaults and requirement constraints picks the domain member.
If not, leave the policy at *Overridable*, the default. The quantity applied by the dynamic default or requirement constraint will be the final configuration quantity for that domain member.
See [Chapter 3, “Understanding Modeling,” Interaction between Default Quantities and Min/Max Settings at Run Time, page 42.](#)
8. If you want the entered (static) quantity to be evaluated against the runtime quantities, designate a new quantity policy:
 - *Min of, Min of f(x)*: Apply the quantity value or expression result as a minimum quantity that will be selected by default for the domain member.
 - *Max of, Max of f(x)*: Apply the quantity value or expression result as the upper limit of the quantity that will be applied to the domain member.
 - *Sum, Sum f(x)*: Add the quantity of each default selection of the domain member with the dynamic defaults and requirement constraints.
For instance, if ItemA has a default of 2 and a dynamic default that specifies the quantity 2, then the runtime quantity of the domain member is 4.
9. Click the quantity policy of the domain member whose quantities you want to edit, and select the policy to apply with the quantity.
10. If you want to set up default quantities that apply for all domain members as a group, select *Use Quantity Policy for All Domain Members* and enter the policy or quantity.
Setting the policy and quantity at the selection point level offers these advantages:
 - It simplifies model maintenance if domain members do not require different quantity settings.
 - If there are no default choices at the domain member level, it allows you to apply static quantities for those domain members that are not defaulted (quantity = 0).
 - If there are default choices at the domain member level, you can set a baseline quantity for all domain members that satisfies a quantity check at the selection point level.
Similarly, click the value in the *Quantity Value/Expression* cell for the domain members) and enter the quantity or select the expression that supplies the quantity for the domain member if it's selected.

Getting Default Selections and Quantities at Runtime Through Attributes

You can specify a static default quantity on each domain member by creating an attribute on its parent class and assigning each of the domain members a value for the attribute. One useful application of this feature is to automate the populating of a bill of materials by using a domain member attribute to set a flag on selected items.

To specify runtime defaults:

1. In the Model Structure view, select the class that contains the domain members for which to set defaults.
2. In the properties table, click the *Add Property* button to add a new row to the table.
3. In the pop-up dialog box, enter a name and data type.
If you want the domain members to have a default value, enter it in the *Value* field. Click *OK*.
4. If you want an external source to supply values for the attribute, change the *Internal* property to *False*.

The SQL property appears. Write the SQL statement to retrieve the desired values.

Warning! Limit attribute values to 10 characters or fewer. Otherwise, the model will not compile.

See [Chapter 5, “Creating Objects for the Model,” Setting Up Binding for External Domain Members, page 75.](#)

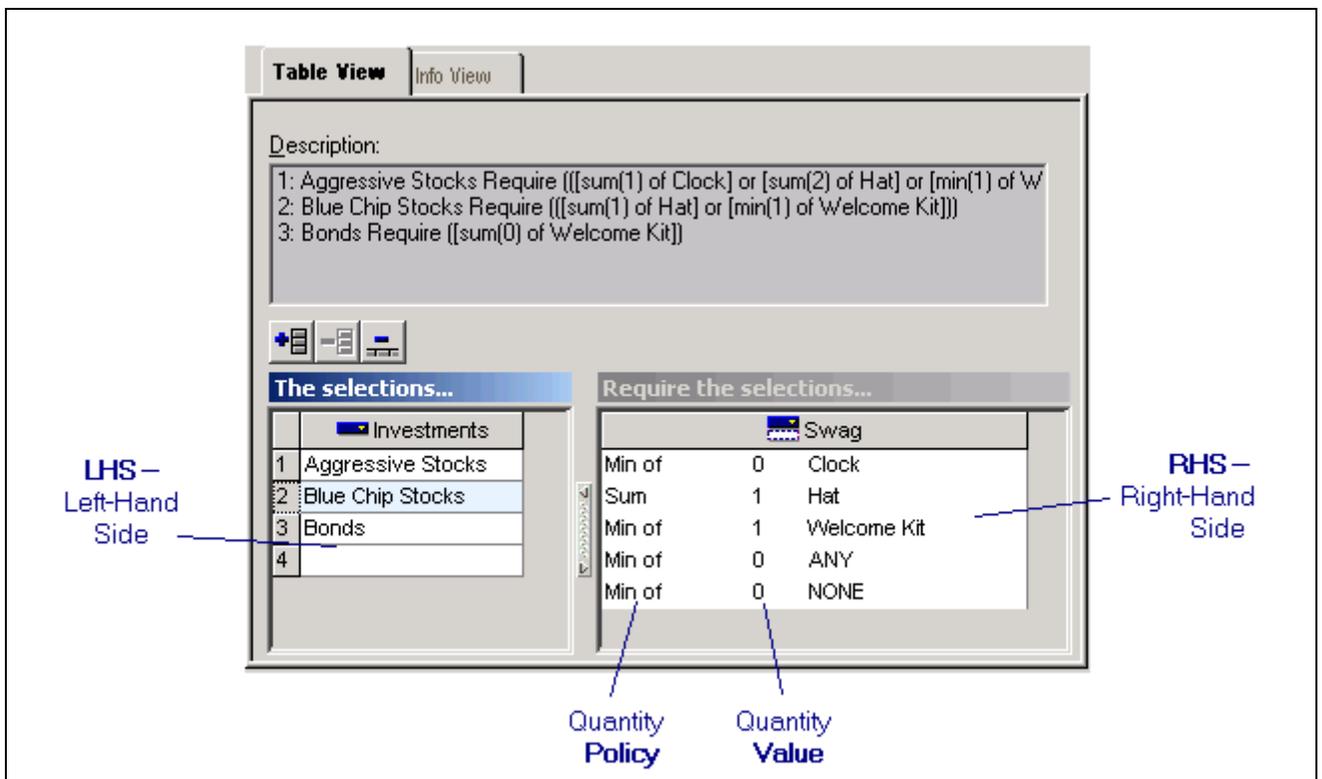
- If you want to assign static values, select each of the domain members in turn and enter the value in the Value column of the attribute that you just created.

Be sure that the value does not exceed 10 characters.

Defining a Dynamic Default Quantity for a Selection

Another way to set up default quantities that are determined at runtime is to write dynamic default relationships with quantities. If you have dynamic default relationships that select by default one or more domain members, you may need to define the quantities in those relationships.

The following example shows default settings that indicate how many of which free gifts to give to a customer based on the investment attribute:



Default settings

Quantity policy settings are:

Quantity Policy	Quantity Values	Resulting Behavior
Min of	0	Default; no selection will be made by the left-hand side (LHS) members.
Min of Min of $f(x)$	$1 - n$ exp_name	No matter how many times the domain member is selected by default by this relationship or another, the dynamic quantity for the domain member will be <i>at least</i> 1 or n .
Max of Max of $f(x)$	$1 - n$ exp_name	No matter how many times the domain member is selected by default by this relationship or another, the dynamic quantity for the domain member will be <i>no more than</i> 1 or n .
Sum Sum $f(x)$	n exp_name	The dynamic quantity for this domain member is the sum of all default selections for this domain member in the session.

To define quantity selection behavior for a dynamic default relationship:

1. Open the relationship editor by double-clicking the name of the relationship on the Components tab. Look for the Quantity Policy and Quantity Value columns in the right-hand side (RHS) of the editor. There should be one of each of these for each selection point in the right-hand side of the relationship. If these columns are not present, then the selection point is not set up for quantity definitions. To enable quantity definition on the relationship, close or minimize the relationship editor. Display the selection point properties on the table editor and set the quantity to *True*. Then reopen the relationship. If you still do not see the quantity columns, check that the relationship binds a class and a selection point rather than a class and a class.
2. If a default selection requires one or more of the selected domain member(s):
 - a. Click on the appropriate LHS row to display the relationship.
 - b. Click the policy of the domain member.
 - c. Select the policy from the drop-down list box.
 - d. Enter the quantity in the value column.

As with other quantity definitions, you can obtain the value from an expression as well.

The resulting dynamic quantity is then subjected to further evaluation against any static default quantities that you may have set in the preceding steps. The result is the final quantity, which is returned for display in the control.

Note. If configuration conditions are such that a default quantity (static or dynamic) would cause a violation, it will not be applied.

Attaching Metadata to Selection Points

To attach metadata to a selection point, you must place an attribute on the selection point. This attribute serves as the source of static default quantities for each domain member in the selection point. The stored metadata can be retrieved by the front end to perform ancillary operations. At runtime, selection point attributes are read-only. Advanced Configurator uses the default value for a selection point attribute if a value is not specified for a selection point attribute.

Float, integer, string, date, and Boolean data types are supported. Selection point attributes can be inherited by subclasses.

Attributes on selection points can be inherited by subclasses but not by domain members; they can't participate in constraints, and they can't be filled from external data using a SQL query.

The selection point attribute is the *On Output* property in the property table.

Note. A selection point attribute and domain member attribute within a class cannot have the same name.

Warning! Limit attribute values to 10 characters or fewer. Otherwise, the model will not compile, generating the following error:

ERROR: Compilation of model <modelname> failed: On GCL class "<classname>", domain member "<domain member name>" has a value for attribute "<attribute name>" of "<value>" which cannot be converted to type `_Integer`.

The following example shows the property table for the class `HardDrives`, with the selection point attribute `AvailDate`:

	Name	Type	Value
	Name	String	HardDrives
	Filename	String	C:\Program Files\Peopl
	Internal	Bool	True
	Descriptio	String	
	ShortNam	String	
	InterfaceTy	String	IDE
	StorageCap	Int	
	Watts	Float	2.00
	AvailDate	Date	5/12/2004

Properties table

To attach metadata to selection points:

1. In the Model Structure view, select the class.
2. In the properties table, add a new row.



Click the Selection Point Attribute button to add a row in the properties table for the selection point attribute.

3. If you want to provide a default value for all domain members in the selection point, enter it in the Value column.

Otherwise, the system uses the value from the input source that you have assigned for the attribute.

4. If you want the default to be different for one or all other selection points of the class, create a separate selection point for each domain member.

Select one of the new selection points in the Model Structure view. The attribute appears in its property table. Enter the desired value in the Value column. Repeat for each selection point for the class.

5. If your application is integrated with Order Capture and you want the attribute's value to be output, enter the selection point attributes on the Output tab of the Schema page the same way as for regular attributes.

PART 3

Product Modeling with Compound Models

Chapter 8
Understanding Compound Modeling

Chapter 9
Working with Compound Models

Chapter 10
Standardizing Compound ModelBuilding

CHAPTER 8

Understanding Compound Modeling

This chapter discusses:

- Applications for compound models.
- Compound model structure styles.
- Architecture.
- Relationships in a compound model.
- Modeling strategy.

Applications for Compound Models

Compound modeling enables you to:

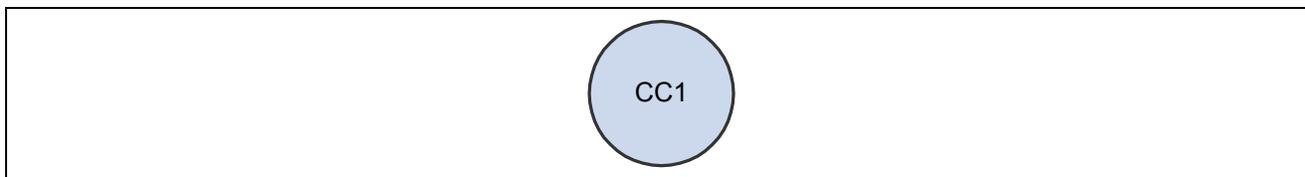
- Create a component model for each configurable component of your product or service offering.
Selection points in each component can receive values from connected components, and can use them in constraints, creating cross-constrained models.
- Specify which configurable components to include in your offering, and how they connect to one another.
- Using PeopleSoft Advanced Configurator APIs and JavaServer Pages, build an interface that lets your user dynamically create, configure, and connect instances of your configurable components.

Compound Model Structure Types

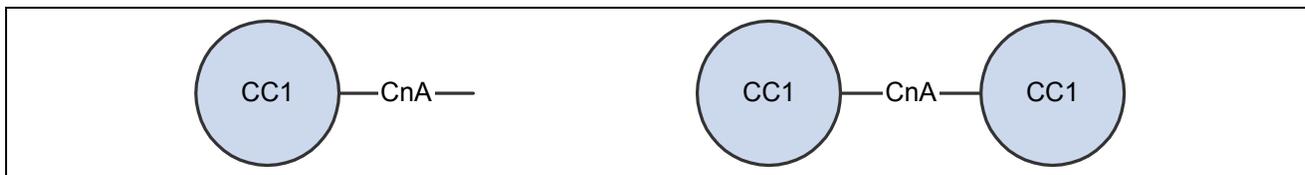
PeopleSoft Advanced Configurator enables a variety of compound model styles as illustrated by the telecommunications scenario of the sample compound model. In the following examples, *CC* indicates a configurable component, and *Cn* indicates a connection.

Note. The sample application illustrates how to create an application that lets a user configure communication services rather than network hardware.

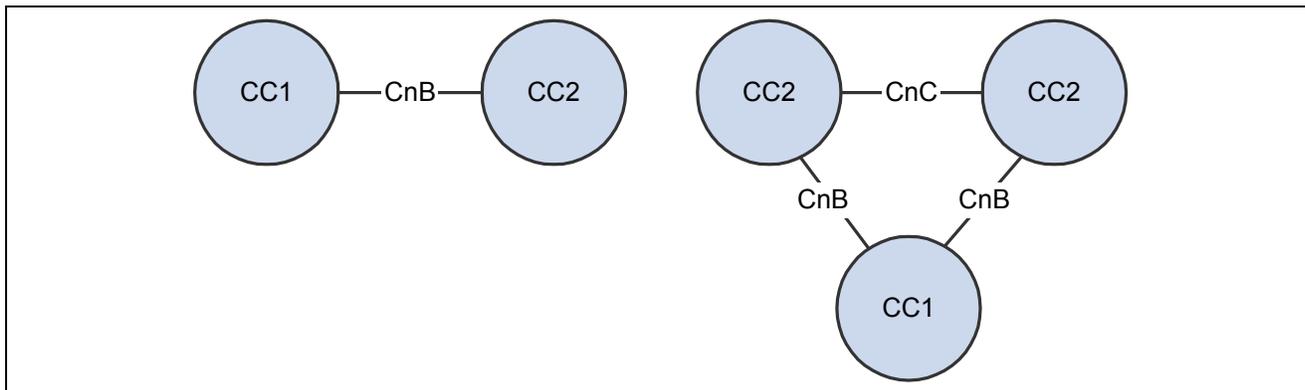
The following diagrams illustrates possible compound model structures, showing configurable components and their relationships (connections):



Configurable component



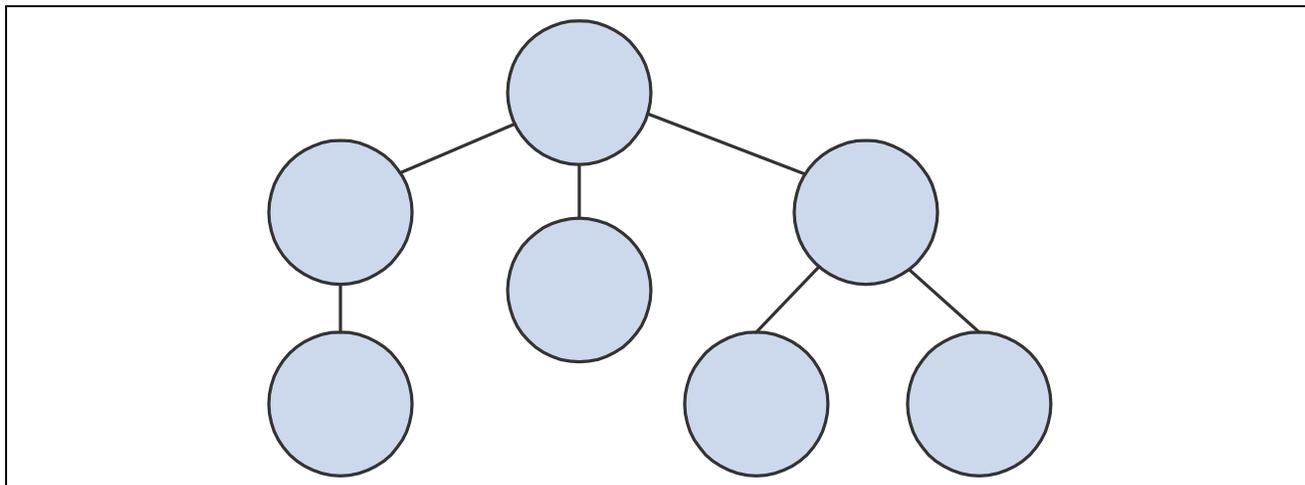
Single configurable component and connection



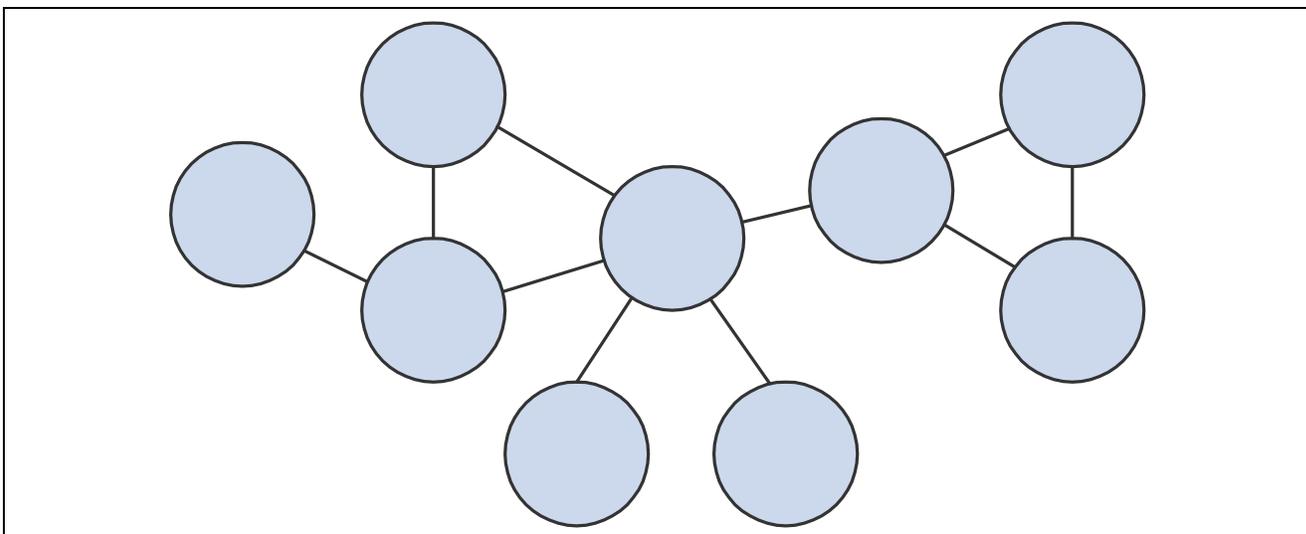
Multiple configurable components and connections

You can template connections by defining them as Connection Types, as CnB is a connection type suitable for joining CC1 and CC2.

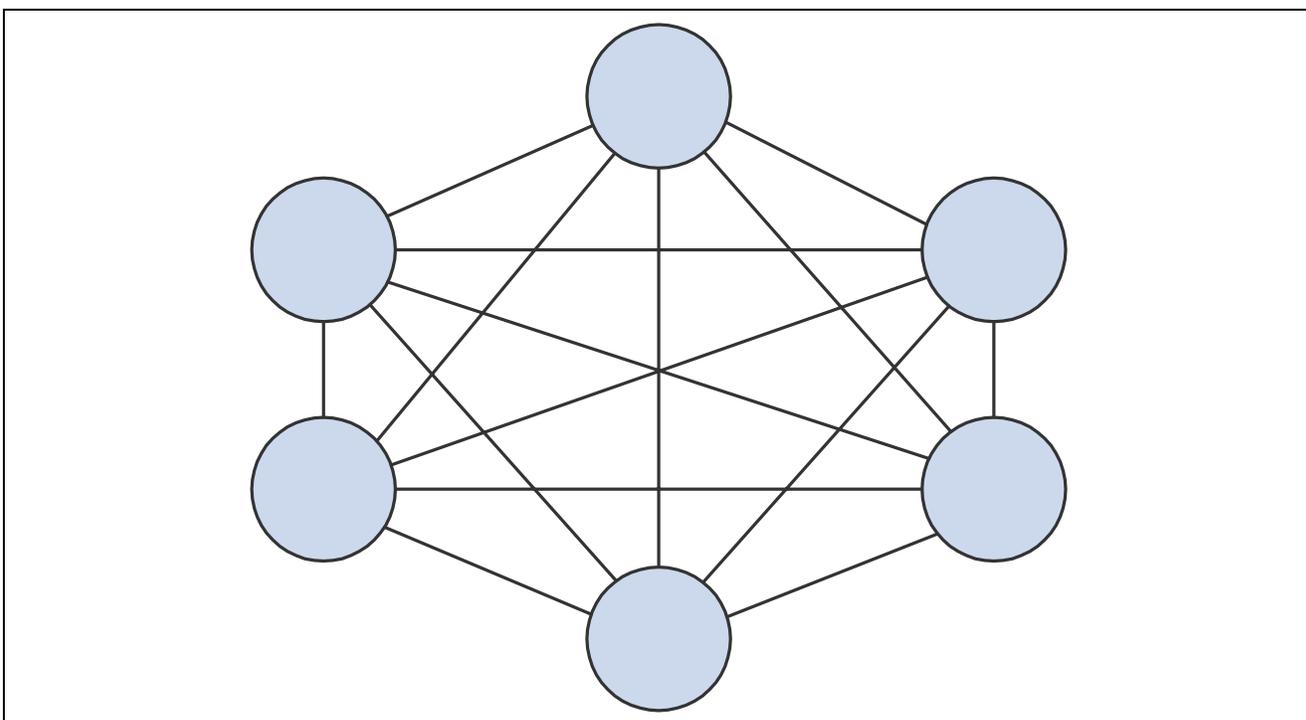
The Configurator provides you the flexibility to structure your compound model in several ways:



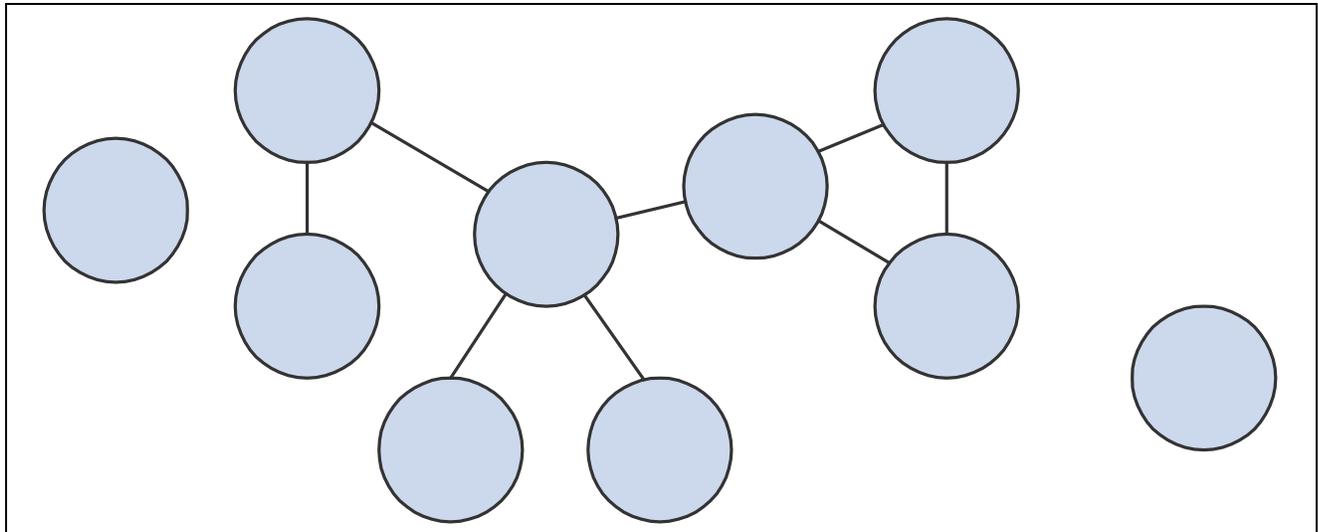
Tree



Network



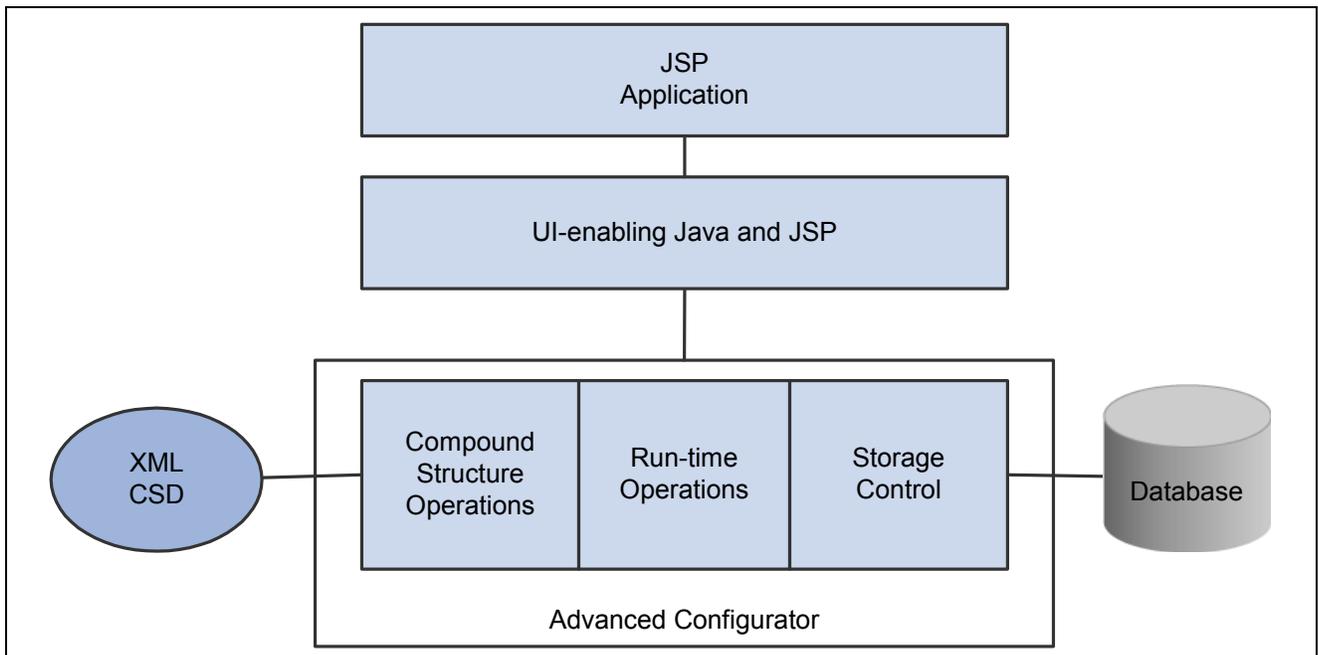
Mesh



Arbitrary

Architecture

Compound structure components extend the Configurator, which runs on the WebLogic application service. The compound structure extensions include Java classes and JavaServer Pages:



Compound model component hierarchy

Some Compound Structure Java classes work behind the scenes to enable you to:

- Create multiple instances of models and configure them.
- Constrain the selection point of one model against the selection point of another.

Other Compound Structure Java classes give you a public API that enables you to create JavaServer Pages that let your user dynamically create, configure, and verify a configuration based on a compound model.

See Also

Part 6, “Building a Custom User Interface,” page 295

Relationships in a Compound Model

Compound models extend the basic precepts of component model objects, and relationships help you define the interaction of the component models. Interactions are defined in the compound model’s Configurable Components.

In a component model, relationships define the behavior of objects at the class and domain-member level. In a compound model, relationships determine how the component models interact. Compound model relationships define connections between components—which components are connected, how many connections are allowed, and what configuration information is sent from one component to others. Using compound model relationships, you can:

- Specify the structure of the compound model—which components must connect with which, and how many.
- Set up conditions for generating valid compound configurations.
- Specify what selection points in components are sources of information, and which are targets for that information.
- Send data from one model to be acted upon by constraints within another model.

Note. The Visual Modeler lets you define relationships; it does not create them so that they are implemented in JSP pages. This must be done by the site developer. By defining compound relationships in Visual Modeler, you provide the validation logic so that an appropriate error message can be generated for the end-user when required relationships are not met.

Compound model relationships address the two central aspects of multiple-model configuration:

- Connected components
- Connection points

Connected Components

Connected components define the structure of the compound model—which and how many component models (components) connect to each other. Minimum and maximum attributes on the relationship specify how many of the specified component. In the Sample compound model, structural relationships establish that Node and Hub components can be connected to Circuit components; Circuit can be connected to Hubs and Nodes; and Hubs and Nodes can’t be connected directly to each other.

Connection Points

Connection points define which data is to be communicated, if any, along the connection. Each connection point defines a specific data element that is to be transferred, and what direction the data flows. This data element can be a user pick or a collection of numeric data across components, or the output of an expression. Important:

Important! The Visual Modeler enables you to define relationships; it does not create them so that they are implemented in JSP pages. This must be done by the site developer. By defining compound relationships in Visual Modeler, you provide the validation logic so that an appropriate error message can be generated for the end-user when required relationships are not met.

Modeling Strategy

PeopleSoft Advanced Configurator provides three approaches to structuring compound models:

- Reference to template component
- Master component type
- Component type with reference

Reference to Template Component

Create a “template” configurable component and use the reference function to create all others. For example:

Template Configurable Component = *ModelPhoneCC*, Min1, Max2

Standard phone component = *StdPhoneCC_refModelPhoneCC*, Min1, Max2

Speaker phone component = *SpkPhoneCC_refModelPhoneCC*, Min1, Max2

...

Master Component Type

Create a component type and base all components on it. For example:

Component Type = *ModelPhoneCT*, Min1, Max2

Standard phone component = *StdPhoneCC_ModelPhoneCT*, Min1, Max2

Speaker phone component = *SpkPhoneCC_ModelPhoneCT*, Min1, Max2

...

Component Type with Reference

Create one component based on a component type, use the Reference function to base others on the original component. For example:

Component Type = *ModelPhoneCT*, Min1, Max2

CC1 = Standard phone component = *StdPhoneCC_ModelPhoneCT*, Min1, Max2

CC2 = Speaker phone component = *SpkPhoneCC_refStdPhoneCC_ModelPhoneCT*, Min1, Max2

...

CHAPTER 9

Working with Compound Models

This chapter discusses how to:

- Get started with compound models.
- Create a compound model project.
- Edit project settings.
- Create a configurable component.
- Delete a configurable component.
- Rearrange component models in the compound model.
- Add and remove a component model from the project.
- Edit default values.
- Create and delete relationships between configurable components.
- Display a compound model relationship.
- Specify required relationships.
- Edit component model versions.
- Compile, run, and test a model.
- Managing simultaneous model development among team members.

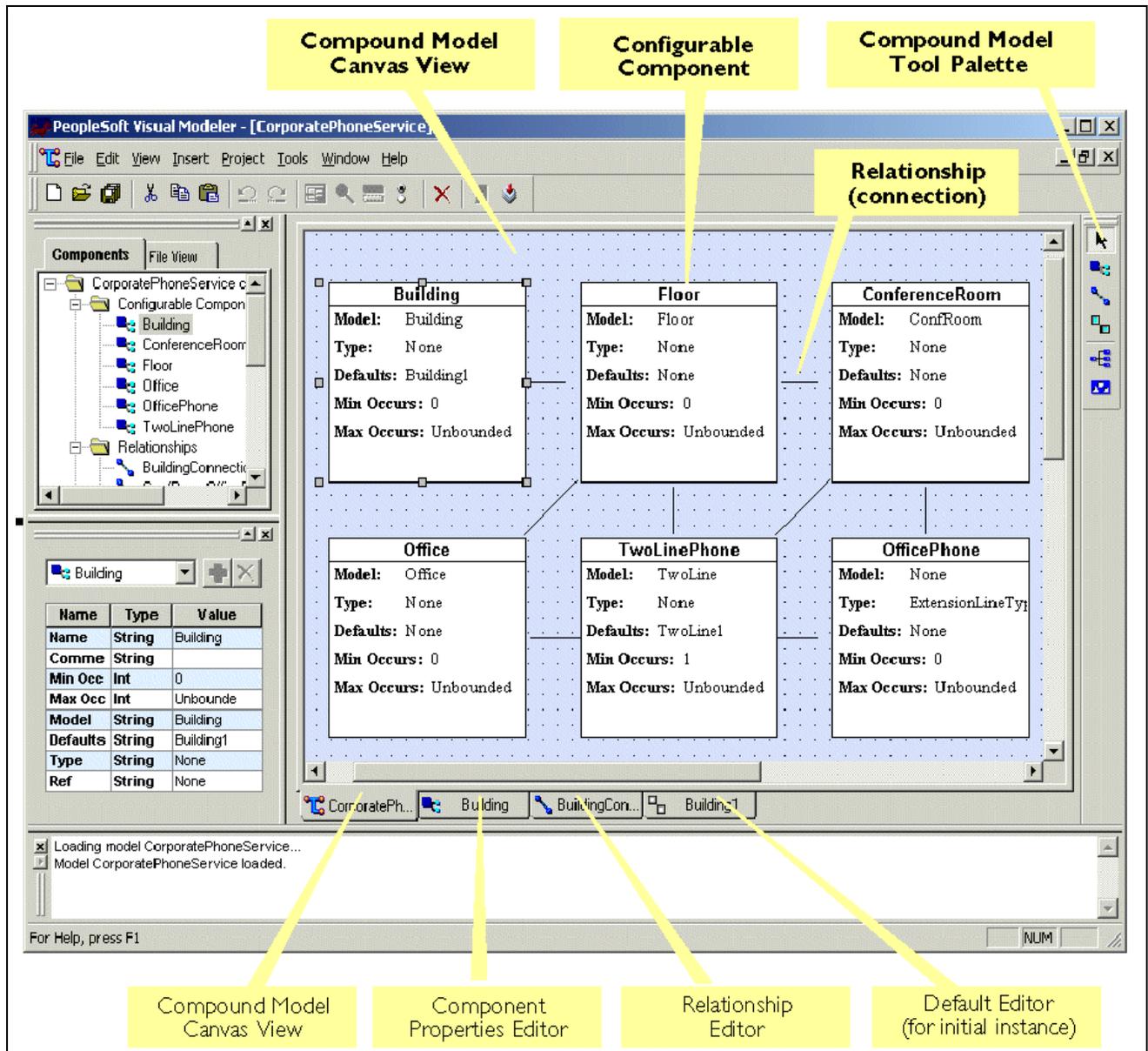
Getting Started with Compound Models

The recommended steps for creating a compound model are:

1. Create the component models that represent the configurable parts of the product.
2. Compile and test the new component models.
3. Create a new compound model in the Visual Modeler workspace, either a new workspace or an existing one (set it as the active project).
4. Insert the configurable components that represent the jump-off points for the separate configurations, and associate each with a component model.
5. Create the relationships between the configurable components.
6. Set the required relationships for each configuration component.
7. Create any defaults for the initial instance of each configuration component.
8. Specify project settings (server name and port).

9. Compile the project and run it on test JavaServer Pages (compound models do not have a Model Tester as component models do).

When you launch a compound model, the Visual Modeler displays the Compound Model Canvas, a UML-style layout grid for constructing and displaying the Compound Model.



Compound model workspace, showing the configurable components of the model CorporatePhoneService



Adds a component to the current project.

Creating a Compound Modeling Project

You can create a new project from the base project template or modify an existing project.

To create a new project from the base project template:

1. Select File, New to display the New dialog box, then click the Projects tab.
2. Specify a project name, storage location, and whether to add the project to a current or new workspace.
When you click OK, a template project tree appears in the Components tab.
3. Add models and files to the project as needed.

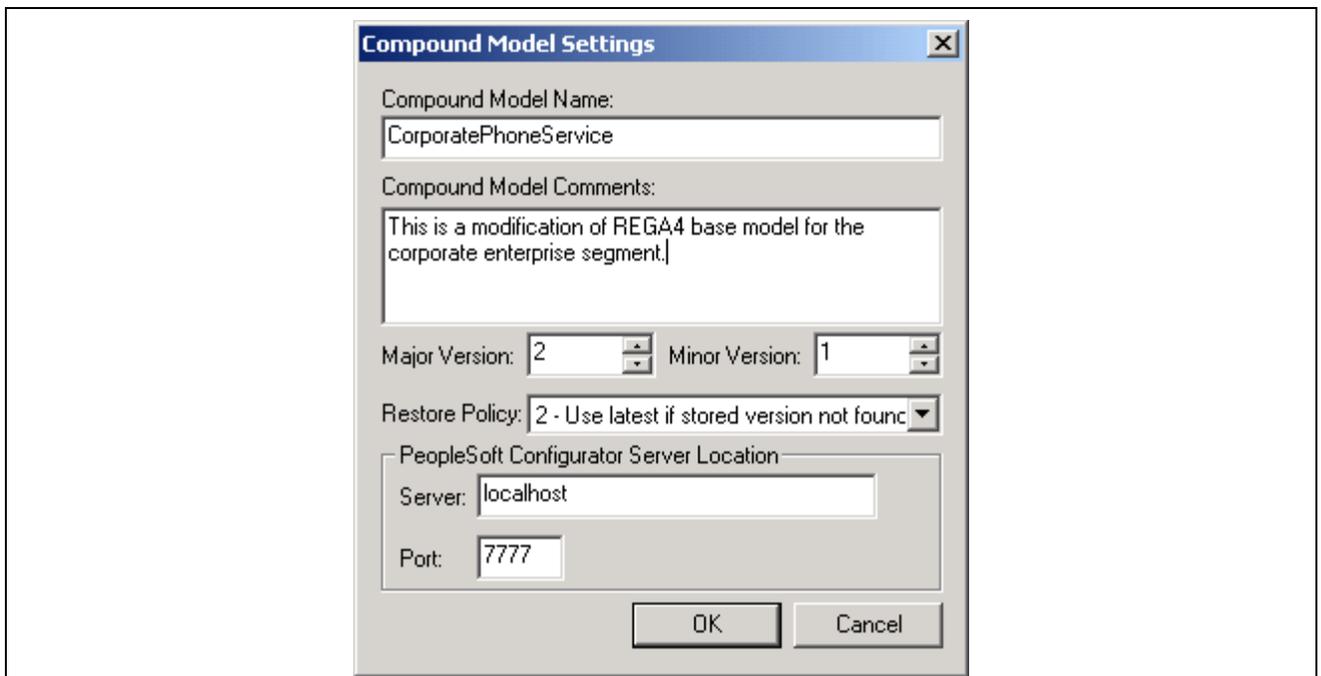
When you need to build a new compound model from an existing one, you can import its structure, relationships, and defaults from its .xml file.

To create a project from an existing one:

1. Create a new compound model Project.
2. Select Project, Import Existing Schema.
A browse dialog box appears.
3. Locate and select the appropriate .XML file.
The imported structure appears in the Components Tab view with the Compound Model canvas.

Editing Project Settings

Properties associated with the compound model are located in the Compound Model Settings dialog box:



Compound Model Settings dialog box

Major Version and Minor Version

Compound model version containing two levels, major and minor, used for model maintenance. It is not to be confused with component model versions.

**PeopleSoft Configurator
Server Location**

Server: The server on which WebLogic and the Configurator are installed.

Port: The port number for the WebLogic application server. The default is 7777.

Creating a Configurable Component

A configurable component is based on an existing component model. Thus, you must create and test each component model before proceeding with this step.

See Part 2, “Product Modeling with a Component Model,” page 9.

Use the component editor to create a configurable component:

Component editor

Name	Identifies the Configurable component whose instance is being acted upon. The name cannot include \, =, <, >, :, ", (, or). The initial character cannot be dollar (\$) or underscore (_), but these characters can be included in other positions. An asterisk (*) cannot be used alone, but it can be used in combination with other characters
Model	Name of the component model to associate the component to. This model must contain the decision point(s) required by the configurable component. Each configurable component references one model. (However, one model can reference more than one configurable component.)
Type	(Optional) The component can be based on a ConfigurableComponentType. The type definition will be used for any data values not specified within the element definition. Specify either <i>reference</i> or <i>type</i> , but not both.
<hr/>	
	Note. If there is a component model associated with the referenced component, it will be overridden by the component model specified for this component.
Reference	(Optional) The component can be based on another configurable component. The referenced component will be used for any data values not specified within the element definition. Either <i>reference</i> or <i>type</i> can be specified, never both.
<hr/>	
	Note. If there is a component model associated with the referenced component, it will be overridden by the component model specified for this component.
Restore Policy	Version of model to use when a stored configuration is requested by an end user. <ul style="list-style-type: none"> • <i>None</i>: The Configurator will use the most recent model version on the Configurator server. • <i>1</i>: Fail if stored version not found.

The Configurator will display an error message.

- 2: Use latest if stored version not found.

The Configurator will look for the version of the model that created the configuration. If it is not available, it will use the most recent model version on the Configurator server.

- 3: Always use latest model.

The Configurator will use the most recent model version on the Configurator server. If that version is not found, it will fail and display an error message.

- 4: Ask if multiple versions.

If more than one version of the model is found, the Configurator will display a choice dialog box requesting that the end user specify which version to use.

- 5: Always use structure version.

Use the version specified in the model's Project Settings.

Max Occurs

A non-negative integer or the term unbounded. Specifies the maximum number of instances that can be created from the component in a single configuration of the compound model.

For example, for a telecommunications product being configured for a moderate-sized business customer, the number of OfficePhones is limited by the number of office setups ordered. You can limit the number of phones the end user can configure by specifying that the Max Occurs value be taken from the OfficeSetup quantity.

Note. PeopleSoft Advanced Configurator does not automatically create components or limit deletion of components based on this number—but it will report that the configuration is invalid if the limit is not met.

- Default = Unbounded
- Minimum value = 1
- Maximum value = Unbounded

Min Occurs

A non-negative integer. Specifies the minimum number of instances that must be created in order to satisfy the requirements of the product model. For instance, if the end user creates an configurable instance of an OfficeSetup, they must also configure at least one OfficePhone for that OfficeSetup. The value of Min Occurs would be 1. A value of 0 would indicate that an OfficePhone is optional.

Note. Advanced Configurator does not automatically create components or limit deletion of components based on this number—but it will report that the configuration is invalid if the limit is not met.

- Default = 0
- Minimum value = 0
- Maximum value is less than Max Occurs

Required Relationships

Any relationship that must be satisfied for the component in order for the configuration to be valid. Important:

Important! This property does not actually implement the relationship; that must be done by the web application developer. It does, however, verify that such a relationship is satisfied.

Setting required relationships for the component here sets up a validation function that, when violated (the end user has not added a necessary component, for instance), an error message is generated for the end user.

See [Chapter 9, “Working with Compound Models,” Creating and Deleting Relationships Between Configurable Components, page 138](#).

Note. “Inherit” check boxes appear when the configurable component has a type or reference specified.

Repeat these steps for each configurable component desired.

To create a configurable component:

1. Make sure that the Compound Model Canvas is displayed, and that the desired project is selected (if the workspace contains more than one project).
2. Do one of the following:
Click the icon at the taskbar on the right of the window.
Or,
Select Insert, Configurable Component.
3. Move the cursor onto the blue grid where you want the configurable component to appear in the model structure.
4. Click once to create the component.
You can drag-and-drop to reposition it.
5. Double-click the component to display its component editor
6. Enter the appropriate values for the elements.
Values for Type, Defaults, and Required Relationships may not be available, as they must be created separately. These can be added later.
7. Repeat these steps for each configurable component desired.

Deleting a Configurable Component

To delete a configurable component in the Compound Model canvas, right-click the desired component and select Delete “<component name>”.

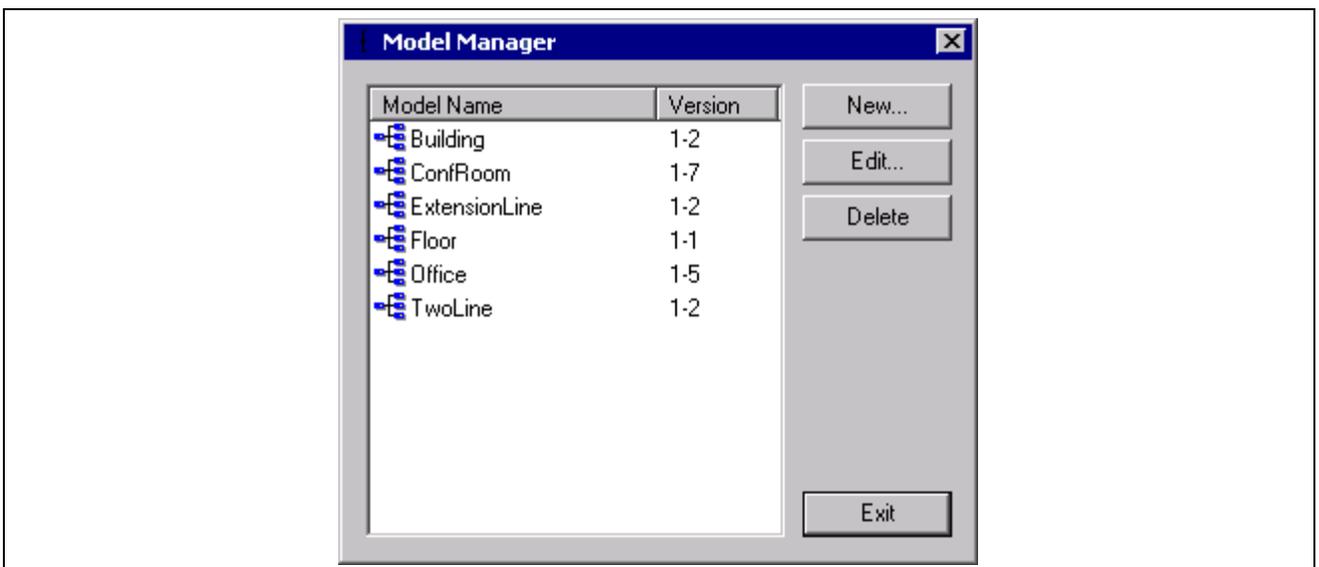
To delete a configurable component in the Components Tab view, right-click the desired component and select Delete from the menu.

Rearranging Component Models in the Compound Model

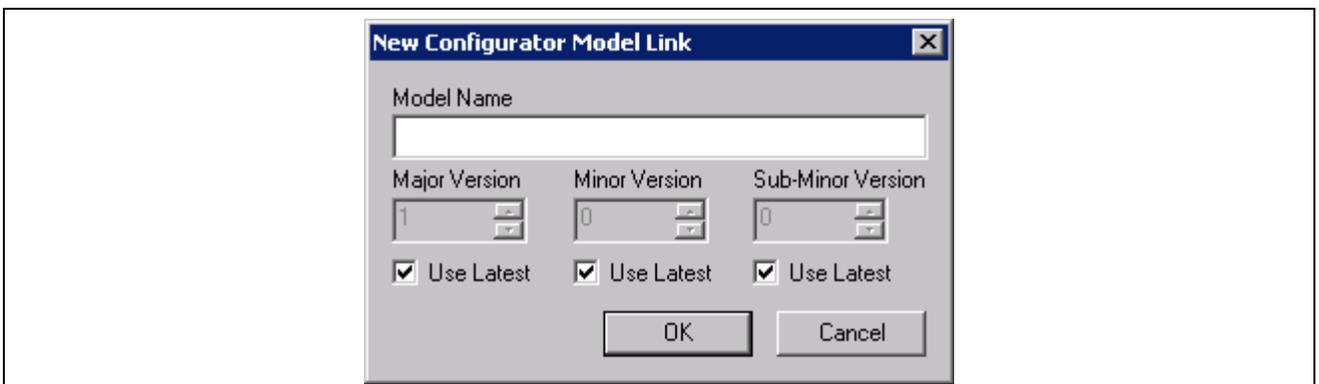
Whenever you are using the Compound Model Canvas to create and link components, you can use the View, Relay Layout Canvas command to efficiently reposition the components in the window.

Adding and Removing a Component Model from the Project

Adding a component model adds a component model to the project. Once added, the model name makes it available for connection, by means of various selection lists, to other component models. Adding a model to the project does not connect it in the compound model. This is done when you create configurable components.



Model Manager dialog box



New Configurator Model Link dialog box



Launches the Model Manager dialog box.

To add a component model:

1. Do one of the following to display the Model Manager:

Select View, Model Manager.

Or,

Click the Model Manager icon in the tool palette.

- In the Model Manager dialog box, click the New button to open the New Configurator Model Link dialog box.
- Enter the model name in the Name field, for example, *ConfRoom* or *Hub*. (The Configurator looks for models at the specified Configurator server location.)

See [Chapter 9, “Working with Compound Models,” Editing Project Settings, page 131](#).

- Set the Version preferences and click OK.

Default for each version level is *Use Latest*, which means the latest version of the model found in the model directory will be used for compiling. You can instead specify a specific model version down to the micro level, or you can specify only certain levels. To specify the version number, clear the check box to enable the version selector.

Examples:

For any version 1.2.1 – 1.2.10,

If you specify	Configurator uses
1-2-5	1.2.5
1-2	1.2.10 (the latest)
1	1.2.10 (the latest)

For any version 1.1.0 – .1.59,

If you specify	Configurator uses
1-1	1.1.59 (the latest)

Removing a component model is more accurately described as removing the link to the component model that exists in the compound model.

Note. Removing a component model from the compound model is not the same operation as deleting a configurable component.

To remove a component model:

- Do one of the following to display the Model Manager:

Select View, Model Manager.

Or,

Click the Model Manager icon in the tool palette.

- In the Model Manager, select the component model that you want to remove from the compound model and click the Delete button.

You will be notified if there are any components that are dependent on the one being removed. Note the dependencies and click OK to the message. You will need to make provision for the removed component model.

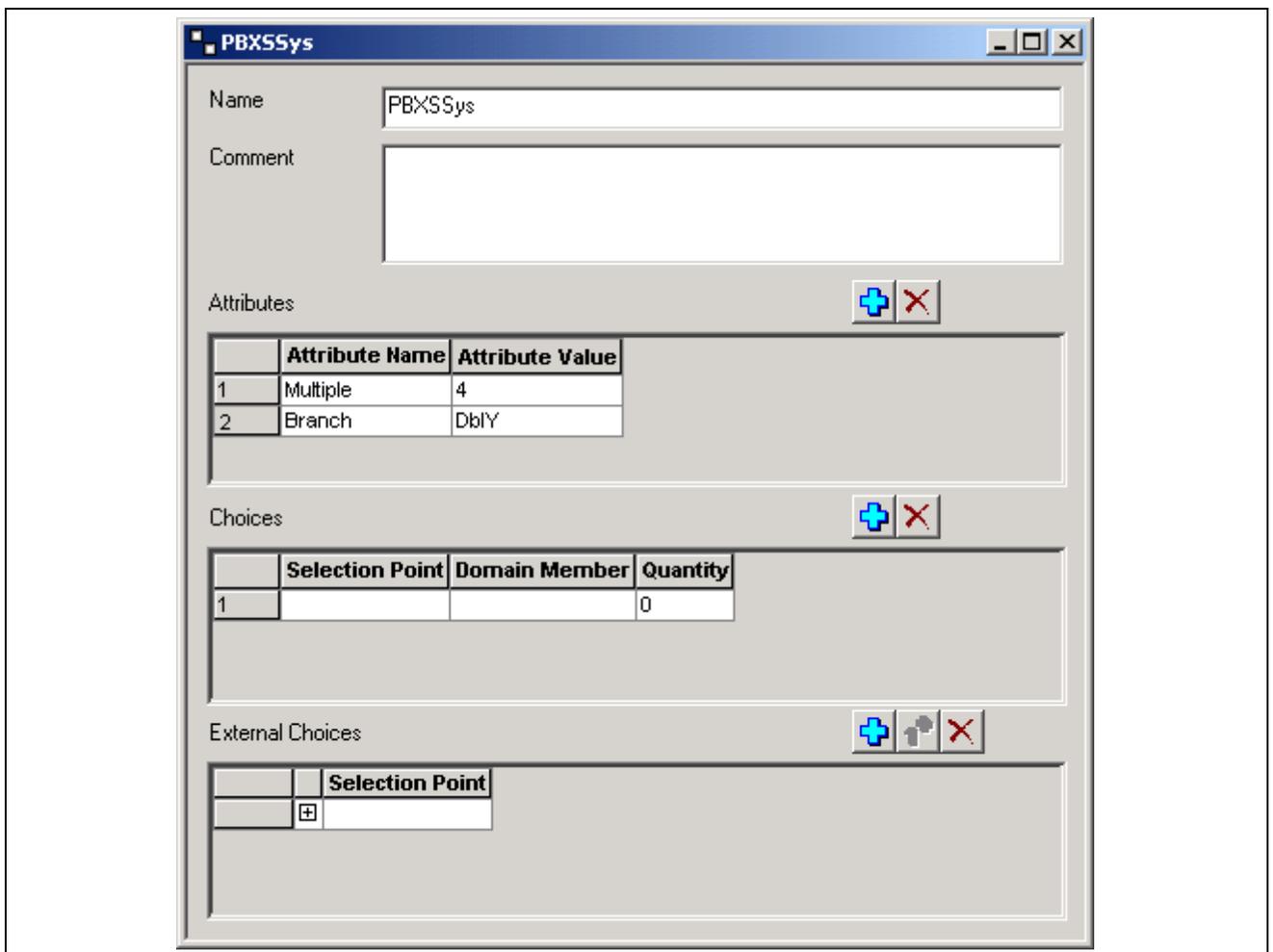
3. Click OK to close the dialog box.

The component will be removed from component windows during the next compile.

Editing Default Values

Defaults are the values that appear on instances of configurable components when they are created from the component models of a compound model. By definition, they are the same as defaults for a component model. But because the initial instance is created within the context of the compound model, with potential dependencies on other component models, their defaults may not be the same as they would be if an instance was created from the model in its standalone state.

Defaults are listed with Configurable Components and Relationships in the Components View tab.



Defaults tab

Attributes

Attribute Name: The name of the attribute in a component model to which to assign the Attribute Value.

Attribute Value: The value that you want the attribute to have when the configurable component is initialized.

	Leave these blank if you do not need attribute defaults.
Choices	A specific value. A domain member of a particular selection point within the associated component model. If applicable, enter a quantity of the item. You can leave this value blank. Use this value when you have a specific selection point in a specific component model to specify.
External Choices	A value provided by a source outside the component model that is only known at runtime, such as user-entered text, numbers, Boolean values, and dates. In the component model, this will be an extern. Leave these blank if you do not need attribute defaults.
	Opens the Defaults window for new defaults specifications.
	Adds rows to the table so that you can edit additional attributes, choices, and external choices.
	On the External Choice element, adds rows so that you can add additional Values to the parent Selection Point.
	Deletes the selected row.

To specify or edit default values:

1. For a new default, open a new Default window in *one* of three ways:
 - To change an existing default double-click its name in the Components View tab.
 - Double-click *Defaults* in the Components tab.
 - From the Configurable Components window, select Insert, Defaults.
 - Click the Defaults icon in the tools palette
2. Specify or edit the desired defaults.
3. In each of the tables in the Default window, use the Add button to add a new attribute, choice, or external choice default.

Creating and Deleting Relationships Between Configurable Components

Relationships in a compound model define interactions between components (component models). By contrast, relationships within component models define interaction between classes and domain members.

Relationship editor

Target

Name of the component model to which data is to be passed.

Connected Components

Component: The component that can be connected to the target component. Click the field to activate the drop-down list box. There can be more than one. Use the Add (+) button to add more components. This relationship property establishes the structure of the compound model and is the basis for required relationship validations. The default is *none*.

See [Chapter 8, “Understanding Compound Modeling,” Modeling Strategy, page 128](#).

Min Occurs: The minimum number of components (instances) that must be connected to the target component for the configuration to be valid.

- A value of 0 means that if the component is created, its connection to the target component is optional (and thus it would not be a Required Relationship).
- A value of 1 indicates that there must be at least one instance of the component connected to the target component.

Max Occurs: The maximum number of components that can be connected to the target component in the configuration. The default value is *unbounded*.

A value of 0 is meaningless. A value of 1 to 99 indicates the upper limit of allowable connections of this component to the target component. *Unbounded* indicates that there is no limit to the number of this component’s connections to the target.



Adds a new connection point.



Opens the properties dialog box for the connected component displayed. Located above the Connection Point Table.



Opens the Relationship editor. Located on the Tools taskbar at the right of the window.



Adds a row (connection point instance) to the Connection Point table.

	Name	Type	Operation	Source	Target
<input type="checkbox"/>	ConnectionPt_CP1	None	Choice		
	Instance	Target	Numeric Data		

An instance of the connection point named Connection PT_CP1

To create a compound model relationship:

1. Do one of the following:

From the main menu, select Insert, Relationship.

Or,

From the Tools taskbar at the right of the window, click the Relationship icon.

The Relationship Editor appears.

2. Enter the appropriate values.
3. To add or view the properties of the connected components, click the Properties button above the Connection Point Table.
4. Set the Connection Point information.

A Connection Point is a data element that is to be passed along the connection represented by the relationship. There can be 0, 1, or more than 1 Connection Points.

5. Click the Add button to add a new Connection Point.
6. Enter Connection Point values.

Type

If you want to assign a Connection Point type, click the field to activate the drop-down list box for selection. Depending on the type you select, the operation and source or Target properties will be provided and will appear in italics (italics indicate properties inherited from types).

If you don't want to assign a Connection Point type, leave the default value of *None* and assign the values Operation and either Source or Target.

Operation

Describes the origin of the data to gather. It can be:

Choice: Indicates the data is a value from a particular selection point domain member or attribute. If *Choice* is selected, you must provide the Target selection point in the Connect Components table (following). Target Variable and Numeric Data do not apply to Choice operations.

Collection: Indicates the data is a set of values taken from all the sources that have a particular attribute in common. If you select the Collection operation, you also must specify Target Variable and Numeric Data (following).

None: Default value; used when an inherited value will provide the value.

Source

Identifies the selection point that provides the data to be communicated over the connection. You may need to open the component model with Visual Modeler (in another window is easier) to obtain the correct name of the selection point.

Target

The name of the object in which to store the collected values of a Collection operation. Not required for Choice operations. This information is located in the component model that is to receive the information (connected component).

7. Configure the instances of the Connection Point by first expanding the row with the Add Instance button.
8. If there is only the header for the instance row as in the image above, click the button to add an instance. If the Connection Point has a type, an instance will already be created and any inherited properties will be entered in the fields. If not, enter these values:

Instance

Specifies how the instances of the Connection Point are to handle the data received. Entries are All, 0, 1, 2, n.

All indicates that all instances of the Connection Point handle the data as specified in the Target DP or Numeric Data columns.

1...n indicates order of creation of the instances of the Connection Point at run time.

Target DP

If the operation is *Choice*, you must specify which selection point (DP) in the source is to receive the data. In this example, the first instance of Ext_voicemailLimitSet will receive the data from the Source DP.

Numeric Data

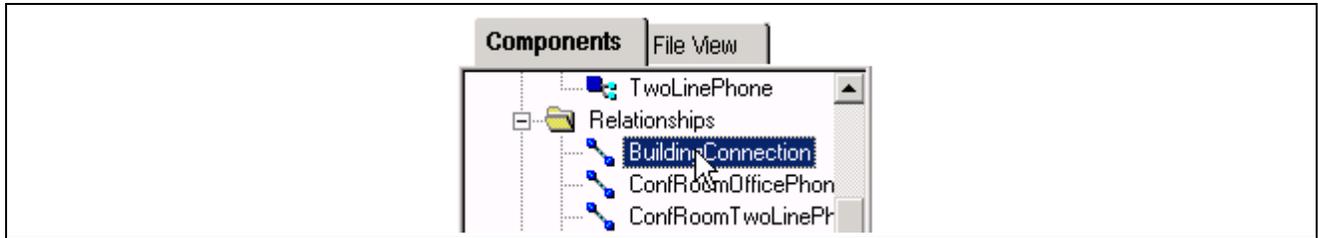
If the operation is *Collection*, you must specify the name of the variable that will contain the data. Because this is a set of numerals, it is called Numeric Data. In this example, all instances of the Source DP will contribute data, which, once all data is collected, will be sent to the object lines.

9. If there are additional Targets to receive source data, click the Add Instance button again to add another row.
10. To add or view documentation about the values, click the Properties button above the Connection Point Table.
11. Click OK to create the new Connection Point.

For the example below, the first instance of <ConnectionPt>_CP1, Instance 1, will send the data to a Target DP called LineA. Instance 2, however, sends its data to LineB; Instance All represents data from both LineA and LineB that the component model sums, placing the result in Target DP LineSum.

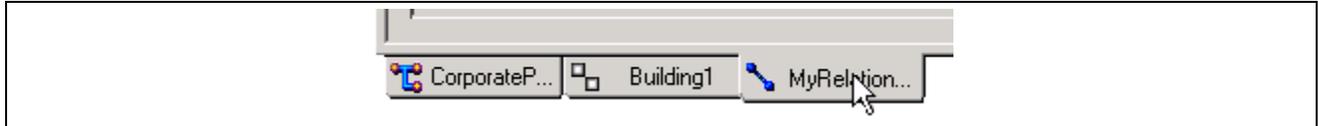
Displaying a Compound Model Relationship

To open a previously unopened Relationship Editor window, double-click the desired relationship in the Components tab.



Opening an existing relationship in the Components tab

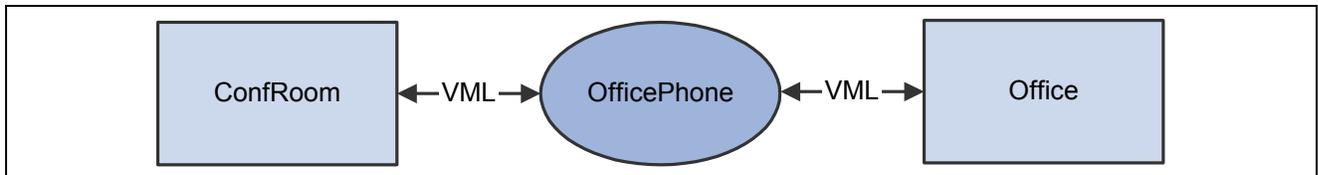
If the relationship is already displayed but is not the active window, click its tab. Relationship windows are indicated by the Relationships icons:



Opening an existing relationship from its tab

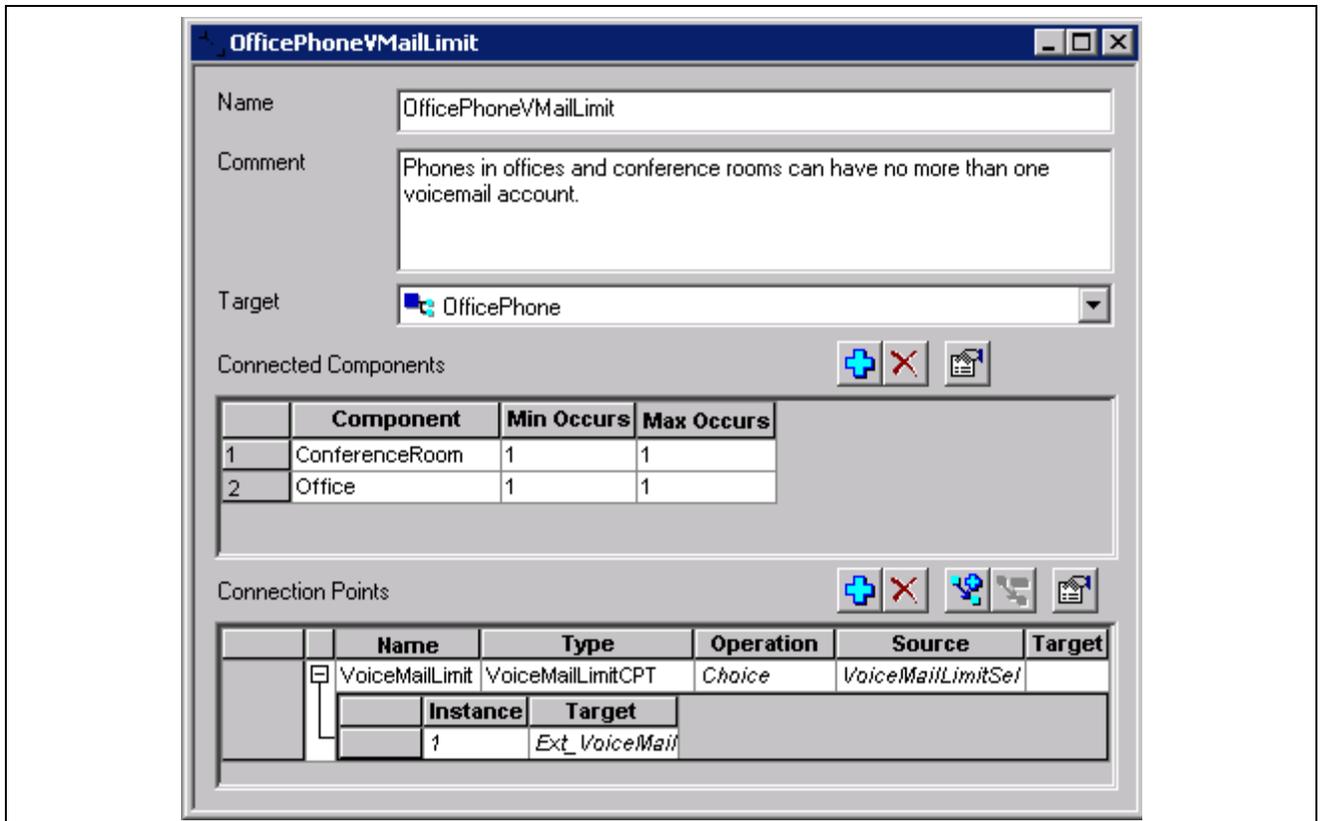
Specifying Required Relationships

Required relationships are those that must be satisfied before a configuration is considered valid. Your product's business logic determines what relationships are required. For example, a telecom service product includes a limit on the number of voicemail accounts on certain phones. The limit applies to phones to be installed in conference rooms and offices. The components ConferenceRoom and Office can then be said to be connected to the target component OfficePhone with a voicemail limit relationship (VML):



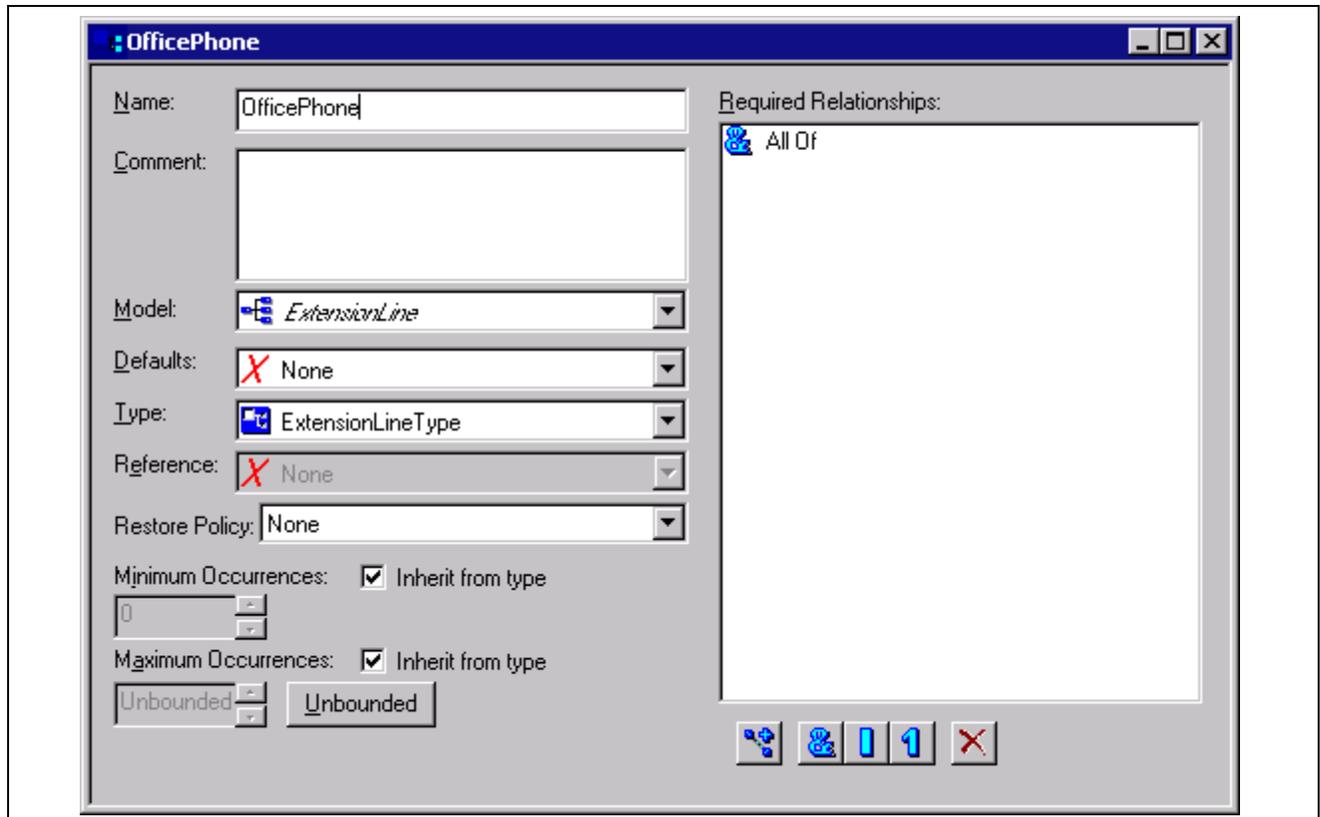
The required relationship VML must connect OfficePhone with ConfRoom or Office

This relationship is represented in the Visual Modeler in the Relationship editor as illustrated below:



Example of a required relationship specification

When a ConferenceRoom is not connected to an OfficePhone (perhaps another phone model is desired), the relationship does not apply. To indicate that the OfficePhoneVMailLimit is required, you must specify it in the target component's properties editor as described in this section.



The component editor showing required relationships



Displays a list of relations for the project.



Click the Add All Of button to insert a blank All Of clause in the Required Relationships panel at the location of the selected clause or relationship name.



Click the Add Any Of button to insert a blank Any Of clause in the Required Relationships panel at the location of the selected clause or relationship name.

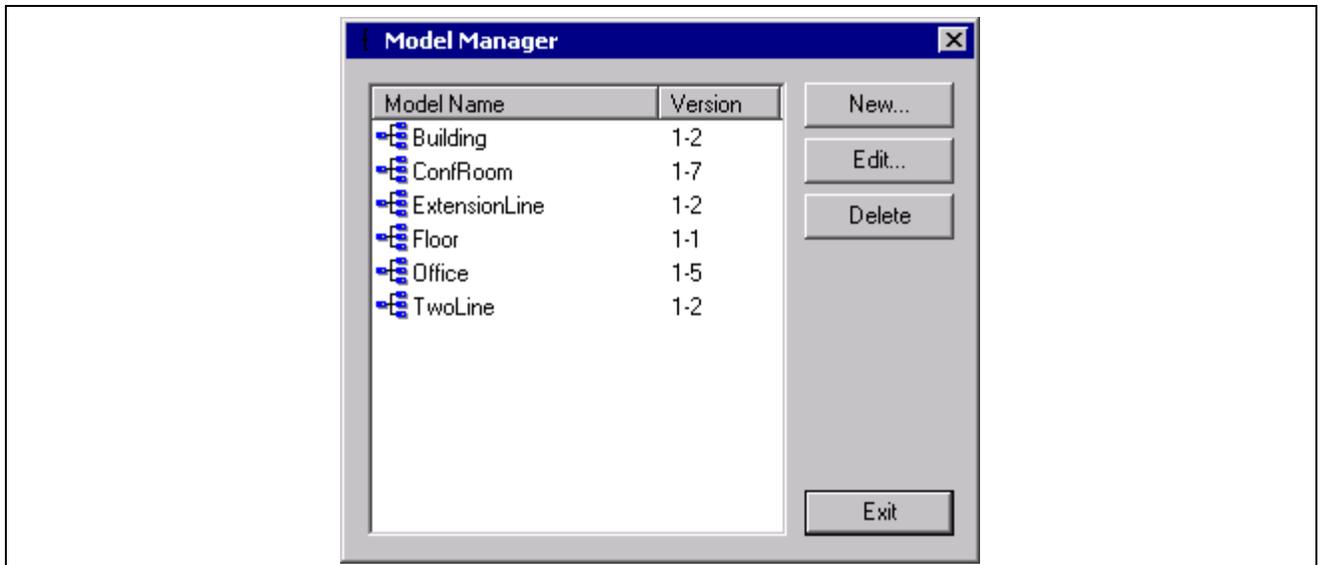


Click the Add Exactly One Of button to insert a blank Exactly One Of clause in the Required Relationships panel at the location of the selected clause or relationship name.

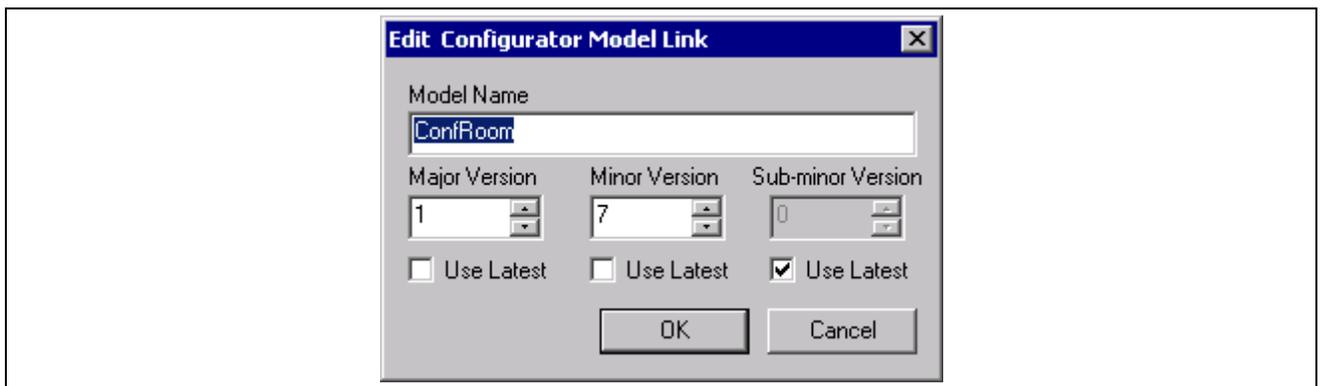
Editing Component Model Versions

At runtime, a compound model communicates with component models (its designated component models), which are located separately. The compound model contains only a link to its component models. Since component models are independently built and maintained, provision for version specification is included in the Compound Model.

Component model versions are specified in the Model Manager dialog box. The default version is 1-0-0 (Major-Minor-Subminor).



The Model Manager dialog box



Editing the compound model version

The figure shows the version of the component model ConfRoom is specified to the Minor level; the Micro level is always the latest. So, at run time, the model accessed will be ConfRoom version 1-7-<latest available>.

To edit a component model version:

1. Do one of the following to access the Model Manager:
Select View, Model Manager.
Or,
Click the Model Manager icon in the tool palette.
The Model Manager appears. Note the version settings for each model.
2. Select the desired component model and click the Edit button.
3. Designate the desired level or select the Use Latest check box.
4. Click OK.

The default for each version level is Use Latest, which means that the latest version of the model found in the model directory will be used for compiling. You can instead specify a specific model version down to the subminor level, or you can specify only certain levels. To specify the version number, clear the check box to enable the version selector.

Compiling, Running, and Testing a Compound Model

A Compound model must be tested on test JSP pages since there is no Model Tester as there is for a component model.

To test a compound model:

1. Compile and run each component model separately in component model mode.
Make sure that each runs to your satisfaction.
2. Select Project, Compile Only to compile the Compound model.
The compound structure definition document (.XML) will be created and placed on the server specified in the compound model settings. You can also launch the Configurator Administration Tool (from the Start/Program menu) to view the .XML file.
See [Part 8, “Advanced Configurator System Administration,” page 411](#).
3. Create JSP pages appropriate to test the connections and constraints between the component models.
JSP pages for the sample compound model are available for modification.
See [Part 6, “Building a Custom User Interface,” page 295](#).
4. Use the JSP pages to create a test web application and deploy it in a test environment.

Warning! Make sure that your browser is cookie-enabled; compound models require the use of cookies to function properly at run time.

Managing Simultaneous Model Development Among Team Members

Not only does Visual Modeler permit simultaneous development of the files of a single model, but team members can work on the different component models simultaneously. Updating a compound model with new versions of component models is managed in the Model Manager, which enables modelers to specify the model version to include in the compound model at compile time. Modelers can specify stable versions of the component models against which to test their updated model, thus controlling their test environment.

CHAPTER 10

Standardizing Compound ModelBuilding

This chapter discusses how to:

- Create and edit configurable component types.
- Create and edit connection point types.

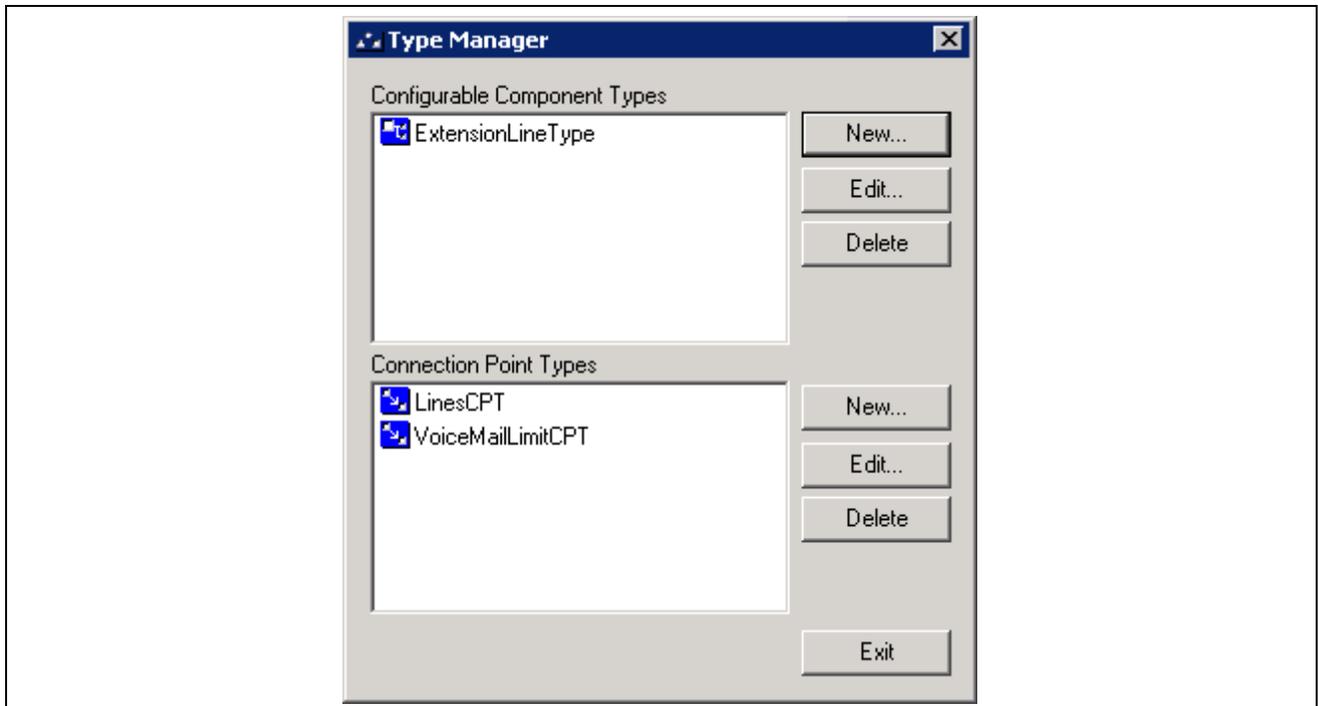
Creating and Editing Configurable Component Types

You can specify the properties of each Configurable Component one-by-one, or, if components share elements, you can create Configurable Component types to use as component templates. Then, when you create a Configurable Component, you can assign it a component type, thereby automatically associating it with a component model, providing it with default values, restore policy, and occurrence limits.

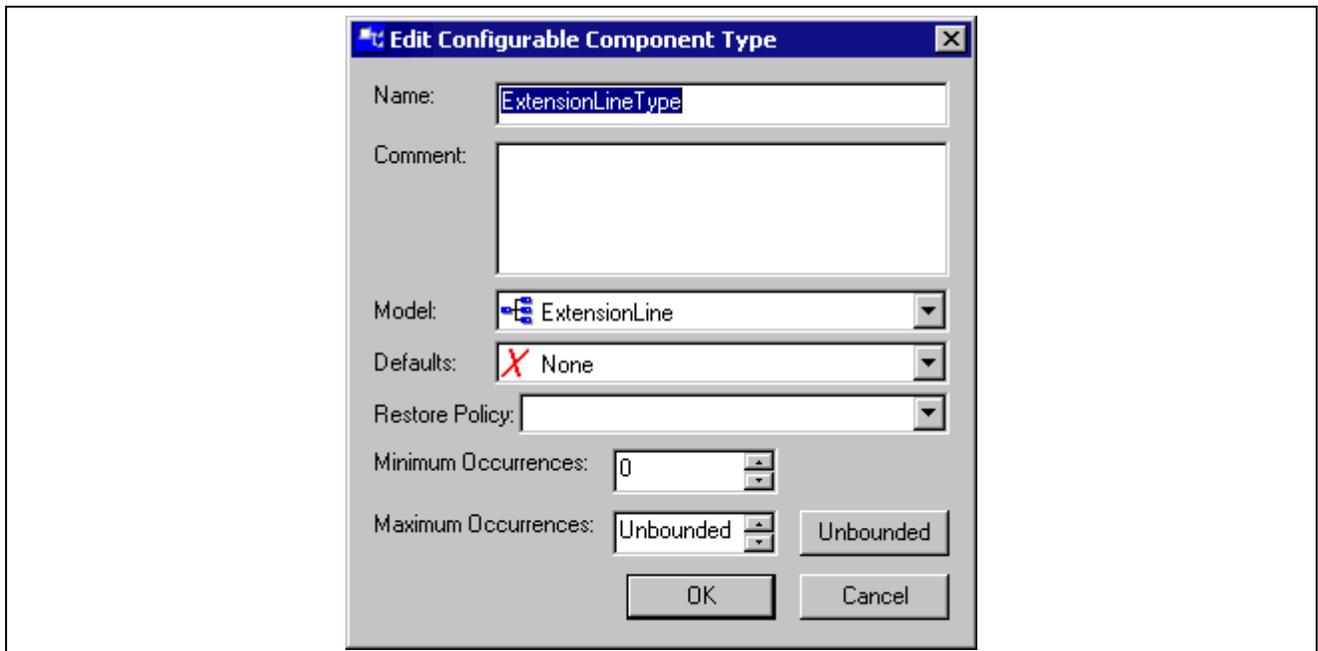
Configurable Component types are assigned in the component editor; you can assign it to a relationship in its properties editor.

See [Chapter 9, “Working with Compound Models,” Creating a Configurable Component, page 132.](#)

Any Configurable Component assigned a component type inherits the type’s properties. In the editor, properties inherited from a type are indicated by italics.



Type Manager dialog



Defining a component type

Name

Identifies the Configurable Component type. The name cannot include \, =, <, >, :, ", (, or). The initial character cannot be dollar (\$) or underscore (_), but these characters can be included in other positions. An asterisk (*) cannot be used alone, but it can be used in combination with other characters.

Model

The component model with which to associate the component.

Defaults

Identifies the default element that specifies the values of the attributes and the domain members when an instance of that configurable component is first created. Defaults will not be available until you describe them.

See [Chapter 9, “Working with Compound Models,” Editing Default Values, page 137.](#)

Note. Compound model defaults override the static defaults set in the component model itself.

Restore Policy

Version of model to use when a stored configuration is requested by an end user.

None: The Configurator will use the most recent model version on the Configurator server.

1 - Fail if stored version not found: The Configurator will display an error message.

2 - Use latest if stored version not found: The Configurator will look for the version of the model that created the configuration. If it is not available, it will use the most recent model version on the Configurator server.

3 - Always use latest model: The Configurator will use the most recent model version on the Configurator server. If that version is not found, it will fail and display an error message.

4 - Ask if multiple versions: If more than one version of the model is found, the Configurator will display a choice dialog requesting that the end-user specify which version to use.

5 - Always use structure version: Use the version specified in the model’s Project Settings.

Minimum Occurrences

A non-negative integer. Specifies the minimum number of instances that must be created in order to satisfy the requirements of the product model. For instance, if the end-user creates an configurable instance of an OfficeSetup, they must also configure at least one OfficePhone for that OfficeSetup. The value of Min Occurs would be 1. A value of 0 would indicate that an OfficePhone is optional.

Default = 0

Minimum value = 0

Maximum value is less than Max Occurs

Note. PeopleSoft Advanced Configurator does not automatically create components or limit deletion of components based on this number—but it will report that the configuration is invalid if the limit is not met.

Maximum Occurrences

A non-negative integer or the term unbounded. Specifies the maximum number of instances that can be created from the component in a single configuration of the compound model.

For example, for a telecommunications product being configured for a moderate-sized business customer, the number of OfficePhones is limited by the number of office setups ordered. You can limit the number of phones

the end-user can configure by specifying that the OfficeSetup quantity value be passed to Max Occurs.

Default = *Unbounded*

Minimum value = *1*

Maximum value = *Unbounded*

Note. PeopleSoft Advanced Configurator does not automatically create components or limit deletion of components based on this number—but it will report that the configuration is invalid if the limit is not met.

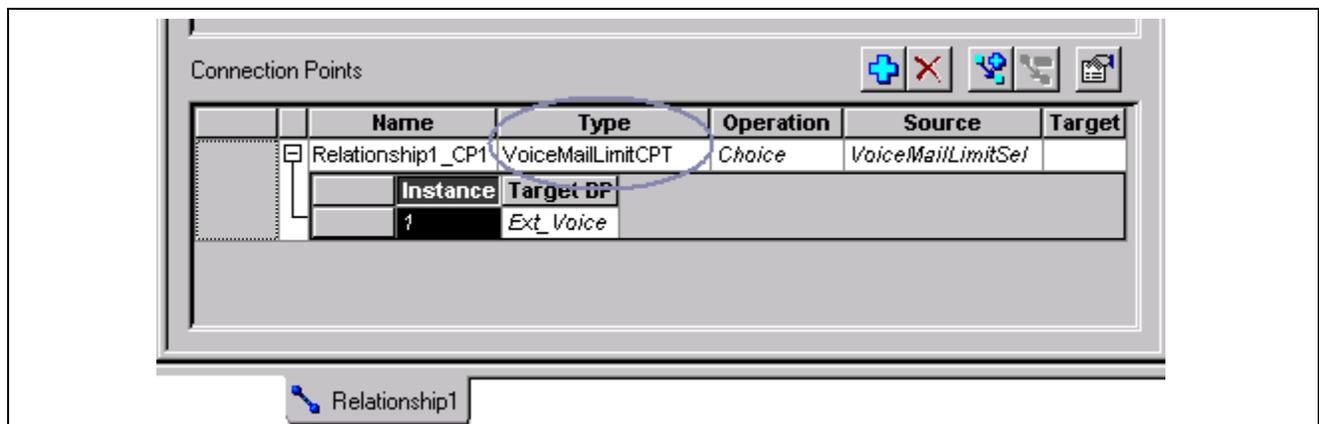
Creating and Editing Connection Point Types

You can specify the properties of each connection between components one-by-one, or, if certain types of connections have common properties, you can create connection point types to use as templates. Then, when you create a compound model relationship, you can assign it a connection point type, thereby automatically providing it with the source object, the data element, and the components that will receive the data.

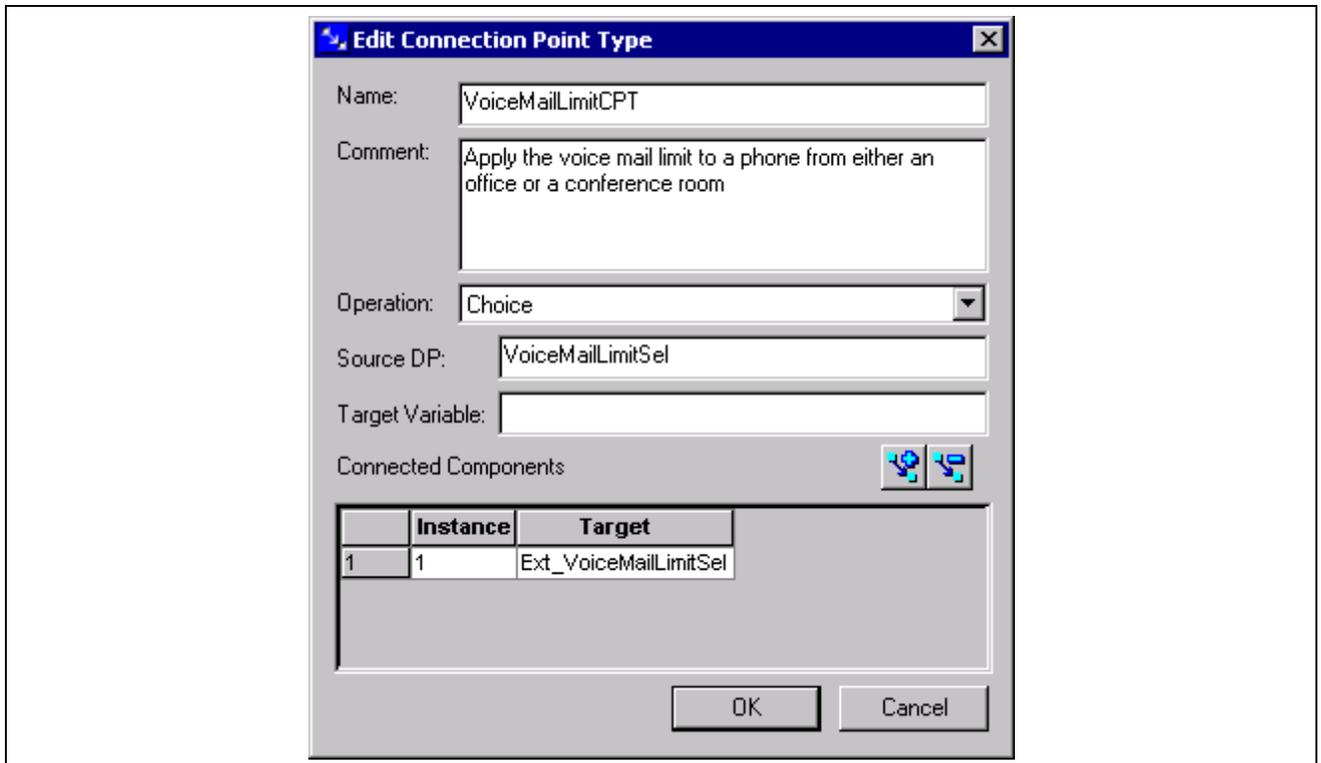
Once you have created a connection point type, you can assign it to a relationship in its properties editor.

See [Chapter 9, “Working with Compound Models,” Creating and Deleting Relationships Between Configurable Components, page 138.](#)

Note. Any connection point assigned a connection point type (when creating a relationship) will inherit the type’s properties. Properties inherited from a type are indicated by italics. This example relationship illustrates how the inherited properties appear in the Connection Points table:



Inherited properties from type VoiceMailLimitCPT are Choice, VoiceMailLimitSel, and Ext_Voice



Editing a connection point type

Operation

Describes the origin of the data to gather. It can be:

Choice: Indicates the data is a value entered by a user. If *Choice* is selected, you must provide the target selection point in the Connect Components table. Leave Target Variable (following) blank, as it does not apply to Choice operations.

Collection: Indicates the data is a set of values taken from all the sources that have a particular attribute in common. If you choose the Collection operation, you also must specify the target variable and numeric data (following).

None: Default value. Use this value when you want the value set for the connection point itself to be the valid value. Otherwise, the Type operation (*Choice* or *Collection*) will override it.

Source

Identifies the selection point that provides the data to be communicated over the connection. You may need to open the component model in Visual Modeler (in another window is easier) to obtain the correct name of the selection point. For Choice operations only.

Target Variable

The name of the object in which to store the collected values of a Collection operation. Not required for Choice operations. This information is located in the component model that is to receive the information (connected component).



Associates a new connected component to the connection point type.



Removes a connected component from the connection point type.

To create a connection point type:

1. Select View, Type Manager.

The Type Manager for both configurable components and connection points appears, listing existing types (if any).

2. If you are creating a new connection point type, click the New button next to its panel.

If you are editing an existing type, select it and click the corresponding Edit button.

The Connection Point Type Editor appears for either case.

3. Enter or edit the values in the editor as desired.
4. Specify connected components by first determining which components are to receive data from the source (source selection point).

If you need to refer to a component model, open it in a separate window of the Visual Modeler.

5. Click the Add button to add a new row (connected component).
6. Enter these values:

Instance	Identifies and specifies which instances of the target DP are to receive data from the source DP. Entries are <i>All, 0, 1, 2, n</i> . <i>All</i> indicates that all instances of connected target (description follows) receive the data. <i>1...n</i> indicates the order in which the target instances were created at run time. For example, the first instance of target, instance 1, will receive the data from the source selection point. You can designate other instances as well.
Target	If the operation is <i>Choice</i> , you must specify which selection point in the source is to receive the data. In an example in which Instance = <i>1</i> and Target = <i>Ext_voicemailLimitSet</i> , the first instance of <i>Ext_voicemailLimitSet</i> will receive the data from the Source selection point.
Numeric Data	If the operation is <i>Collection</i> , you must specify the name of the variable that will contain the data. Because this is always a set of numerals, it is called numeric data. In an example where Instance = <i>All</i> and Numeric Data = <i>lines</i> , all instances of the source selection point will contribute data, which, once all data is collected, will be sent to the object lines.

7. If there are additional targets to receive source data, click the button again to add another row.
8. Click OK to create the new type.

New connection point types will hereafter appear for selection in controls in the properties editor of the component.

PART 4

Application Extensions

Chapter 11
Client Operations Processor API

Chapter 12
Using the COP Java API

Chapter 13
Understanding the Configurator XML Interface

Chapter 14
Retrieving Model Information

Chapter 15
Updating a Configuration

Chapter 16
Retrieving Configuration Information

Chapter 17
Retrieving Saved Configuration Information

Chapter 18
Copying a Configuration

Chapter 19
Using Batch Configuration Mode

Chapter 20
Changing the Order Status of a Configuration

CHAPTER 11

Client Operations Processor API

This chapter discusses the PeopleSoft Configurator Client Operations Processor (COP) Java API and its application classes.

Understanding the COP Java API

The COP Client Operations Processor provides, through its Java API, the public interface to the Configurator. Your User Interface (UI), or other application, calls the COP Java API to communicate with the Configurator Engine and its associated modules.

In many cases, developers designing a UI to use with the PeopleSoft Configurator will not have to make COP Java API calls (or write any code) themselves. Instead, they can use the Configurator Control Templates, which allow them to use standard Configurator controls to present their interface. The Control Templates are used within JavaServer Pages. The Control Templates themselves make COP Java API calls to implement the behavior of the standard controls.

However, in some cases, you may want to have controls whose behavior or appearance is different from any of the existing Configurator controls. In these cases you will have to modify the code in the JavaServer Pages (JSP), or write your own from scratch. Alternatively, you may need to create a UI that uses something other than JSP or standard Web technologies in general.

In that case, these chapters are the reference you need. They describe how to understand, modify, and write Java code that communicates with the Configurator by making COP API calls.

You can use the COP API to:

- Connect to a particular model.
- Create objects that represent user choices.
- Submit these choices to the Configurator Engine, and get back the results.
- Get user-readable display information, such as domain member attributes and text descriptions of violations.
- Get delta-pricing information.
- Get configuration delta information.
- Verify a configuration.
- Save and restore a configuration.

Choices

Choices are inputs to the model that specify values. Usually they are user inputs (set through a UI), but they can also be programatically generated. The Configurator has two kinds of choices: extern variables and domain members.

Extern variables (also called simply “externs”) are named variables whose values are of types int, double, string, boolean, and date. The UI may restrict what a user can enter so that, for example, the extern variable only contains a single floating-point number. Extern variables are frequently used in expressions.

Domain members are individual, discrete choices. They are arranged into groups, each group associated with a decision point (or selection point). A domain member usually has a number of attribute values—such as its description, size, or color—that the Advanced Configurator can access and use in various ways.

Decision Points and Domain Members

Two modeling concepts—decision points and domain members—are very important in understanding how to use the COP Java API. This section describes these concepts and how they relate to the UI.

The UI is a visual representation of your model (or, at least, a portion of the model). The model contains decision points and domain members, some of which will be displayed in the UI.

The figure shows a sample UI for a “Sandwich Model,” in which the user can order a sandwich by choosing the filling, bread, condiments, extras, and temperature.

The image shows a web-based user interface for a "Sandwich Model". At the top, there are two buttons: "Submit" and "Reset Model". Below these are four panels, each representing a different category of choices:

- Filling:** Radio buttons for Roast Beef, Tuna (selected), Chicken, and Veggie.
- Bread:** Radio buttons for White, Rye, Croissant, French Roll, and Wheat (selected).
- Condiments:** Checkboxes for Cranberry Relish, Horseradish, Mayonnaise, Mustard, Pepper (checked), and Salt.
- Temperature:** Radio buttons for Hot and Cold (selected).

Sandwich model UI

A decision point is a collection of associated options that the user (or, through software, the application) can choose from. The options in a decision point are usually alternatives, although they need not be mutually exclusive. The individual options themselves are called domain members of that decision point. For example, in the Sandwich Model, the area labeled “Filling” represents a decision point—the user can select a sandwich meat from this collection. Each of the possible choices—Roast Beef, Turkey, Chicken, Tuna—represents a domain member of this decision point.

The COP Java API does not directly define or use decision point objects or domain member objects. Instead it encapsulates them in objects of type `ControlData` (for decision points) and `ControlItem` (for domain members). Frequently, in discussing the COP Java API, these distinctions will be blurred when there’s no chance of ambiguity.

A decision point has certain properties—such as a name or a “multiselect” property—that can be obtained (through Java COP API calls) from the corresponding `ControlData` object. Likewise, a domain member has properties that can be obtained from the corresponding `ControlItem` object. But `ControlData` and `ControlItem` objects can also contain additional information, often representing current or developer-determined conditions. (For example, the `ControlItem` objects belonging to a `ControlData` object may have a sort-ordering established by the software when it created the `ControlData` object. This ordering is not a part of the corresponding decision point and its domain members.)

See Also

[Chapter 12, “Using the COP Java API,” `ControlData`, page 171](#)

[Chapter 12, “Using the COP Java API,” `ControlItem`, page 176](#)

Application Classes

This section lists the COP Java API classes and the primary uses of each.

ClientOperations

`ClientOperations` is the principal class you will be using. Its methods include:

- Initializing the session (connecting to a model).
- Creating objects that represent user choices (user selections, user eliminations, and extern variables).
- Submitting the user choices to the Configurator Engine that is retrieving the results (`ControlData` objects, delta-pricing information, numeric data, violations).
- Retrieving the names and values of extern variables.
- Restoring a configuration, getting the current configuration.
- Other actions.

`getControlData` is arguably the most important method in the `ClientOperations` class. It creates a `ControlData` object representing a decision point, which in turn contains an array of `ControlItem` objects representing the decision point’s domain members. The UI obtains most of the information it needs from these objects.

Configuration

The Configuration class represents a configuration of choices (which may be user, computer, default choices, and extern variables), together with the configuration attributes and the model's name, version, and compileID. The ClientOperations class has methods to create a Configuration object that represents the current configuration, and to restore a configuration represented by a Configuration object.

The Configuration class includes methods for:

- Writing a representation of itself in XML format.
- Reading configuration data that has been written in XML format.
- Getting the Configuration object's data (choices, configuration attributes, model information).
- Setting the Configuration object's data (choices, configuration attributes, model information) directly.

ControlData

The ControlData class represents a decision point, and contains display information for the decision point and its domain members (ControlItem objects). ControlData objects are created by the ClientOperations method getControlData. The ControlData class includes methods for:

- Getting all ControlItem objects in this ControlData object.
- Getting iterators that sort and filter the ControlItem objects in specified ways.
- Getting current configuration values of the decision point (state, choices, quantity, and violations).
- Getting properties of the associated decision point (name, multiselect, optional, supports quantity).

ControlItem

The ControlItem class represents a domain member, and contains display information on the domain member and its attributes. ControlItem objects are obtained from a ControlData object that represents the domain member's decision point. The ControlItem class includes methods for:

- Getting display attributes and delta-pricing value for the domain member.
- Getting current configuration values of the domain member (state, elimination level, validity, and violations).

Choice

Choice is the superclass for DMChoice (for domain member choices) and EVChoice (for extern variable choices). You pass a vector of Choice objects (possibly including both DMChoice objects and EVChoice objects), one for each user choice, to the Configurator in the ClientOperations method processChoices, which then processes the choices to create a solution state.

In previous versions of the Configurator, objects of type Choice were used to represent domain member choices. For compatibility, the current Choice class still retains a number of methods that only make sense for domain member choices, but these methods are deprecated, and identical methods have been included in the DMChoice class. Use the methods in the DMChoice class instead of the deprecated methods in the Choice class.

The Choice class also contains non-deprecated methods for a few actions that are meaningful for both DMChoice and EVChoice, including:

- Getting the XML tag for this choice.

- Determining if the choice is a user selection.
- Getting the name of the decision point or extern variable associated with the choice. (The method for this is called `getDecisionPointName` for backward compatibility.)

DMChoice

DMChoice is a subclass of Choice.

The DMChoice class is used to represent domain member choices (selections and eliminations). You create a DMChoice object by calling the ClientOperations method `makeSelectedChoice` (when the user wishes to include a domain member) or `makeEliminatedChoice` (when the user wishes to exclude a domain member).

The DMChoice objects generated by `makeSelectedChoice` and `makeEliminatedChoice` represent user choices. However, by using the DMChoice method `setState`, you can change them to represent computer or default choices (or various combinations).

EVChoice

EVChoice is a subclass of Choice.

The EVChoice class is used to represent extern variable choices. You create an EVChoice object by calling the ClientOperations method `makeExternVarChoice`.

ItemFilter

The ItemFilter class specifies a filter for a decision point's domain members. You can use it to filter out all eliminated domain members, or all eliminated domain members whose elimination values fall outside some given range.

The ClientOperations method `getControlData` has an ItemFilter parameter, which you can use to filter out domain members that, in some models, the UI designer does not wish the user to see (such as all eliminated items). The filter parameter is optional. If you supply null instead, no filtering will be done. The exception is discarded domain members, which are always filtered out.

See [Chapter 12, “Using the COP Java API,” Handling Deleted Domain Members, page 176](#).

ItemIterator

The ItemIterator class is used to iterate through the domain members of a decision point—more specifically, through the `ControlItem` array of a `ControlData` object. An iterator (object of type `ItemIterator`) returns the `ControlItem` objects both sorted and filtered, as specified by your software. (The `ControlItem` array itself is neither sorted nor filtered by the COP.)

The sorting done by an iterator can either be a standard sort order implemented by the COP or a custom sort order determined by a comparator you provide. You can also have more than one iterator for a single `ControlData` object.

See Also

[Chapter 12, “Using the COP Java API,” Sorting and Filtering, page 174](#)

ExternVar

The ExternVar class represents an extern variable, and gives access to its name, type, and value.

NumericData

The NumericData class is used to get the type and value of numeric variables. Some models contain variables that you may wish to display in the UI, such as "total grams of saturated fat" or "number of video card slots remaining." These values are generated by the Configurator Engine, based on formulas specified in the model. From the point of view of the COP, they are read-only.

Numeric data are distinct from: configuration attributes; extern variables; and domain member attributes, prices, and quantities.

Violation

The Violation class is used to report on violations associated with the configuration, decision points, and domain members. It returns a user-readable explanation for the violation.

CHAPTER 12

Using the COP Java API

This chapter describes how to use the COP Java API to accomplish tasks required by the UI, including the following:

- ClientOperations.
- Initializing the COP.
- Processing and displaying a page.
- Saving and restoring a configuration.

ClientOperations

This is the primary class you will use. It includes methods for initializing a session, creating objects to represent user choices, submitting user choices to the Configurator Engine, getting back the results in the form of displayable information, and ending a session.

Note. Several ClientOperations methods, such as `getDecisionPointNames`, make a distinction between “public” decision points and “all” (public and private) decision points. The current version of the Configurator does not implement this distinction; all decision points are considered public.

Methods

The following methods are contained in the ClientOperations class:

```
void release()
void initialize(Locale appLocale, String modelName, String modelVersion, String⇒
    compileID, boolean needHtmlEncoding)
void initialize(Locale appLocale, String modelName, String modelVersion, boolean⇒
    needHtmlEncoding)
Configuration getConfiguration()
String[] getDecisionPointNames(boolean allObjects)
String[] getNumericDataNames()
String[] getExternVarNames()
ControlData getControl(String objectName)
ControlData getControlData(String objectName, String[] attributes, boolean sort,⇒
    ItemFilter filter)
ControlData getControlData(String objectName, String attribute, boolean sort, Item⇒
    Filter filter)
NumericData getNumericData(String objectName)
ExternVar getExternVar(String objectName)
double getTotalPrice()
```

```

void processChoices(Vector choices)
Violation[] getViolations()
boolean verifyConfiguration(Vector choices)
boolean verifyConfiguration(Configuration config)
DMChoice makeSelectedChoice(String dpName, String dmName, double qty)
DMChoice makeEliminatedChoice(String dpName, String dmName)
EVChoice makeExternVarChoice(String name, Collection values)
String getModelVersion()
String getModelCompileVersion()
Locale getLocale()
ffBaseBillOfMaterials getBOM()
String[] getCompileVersions(String modelName, String modelVersion)
String getModelName()
String[] getModelNames()
String[] getModelVersions(String modelName)
int restore(Configuration config, Locale appLocale, boolean needHtmlEncoding, int⇒
    policy)
void setPricingData(Map dpNamesAndAttributes)
String[] getIncompleteDecisionPointNames()
String[] getViolatedConstraintNames()
String[] getViolatedDecisionPointNames()
Date getSolveDate()
void setSolveDate(Date solveDate)

```

Initializing the COP

Before calling any other ClientOperations methods, you must attach the COP to a particular model to initialize your session. Do this by calling the ClientOperations methods initialize or restore. There are two different forms of initialize:

```

void initialize(Locale appLocale, String modelName, String modelVersion, boolean⇒
    needHtmlEncoding)
void initialize(Locale appLocale, String modelName, String modelVersion, String⇒
    compileID, boolean needHtmlEncoding)
void restore(Configuration config, Locale appLocale, boolean needHtmlEncoding, int⇒
    policy)

```

Use the first form when you have a model name and version, and wish to use the most recently compiled version of the model. In this case, you simply call the first form of initialize with the indicated parameters.

The second form is used when there is, in addition to the model name and version, a specific compilation version (indicated by compileID) of the model that you wish to use. In this case, call the second form of initialize with the indicated parameters, including the compileID.

There is one form of restore. It is used when you wish to load a previously saved configuration when initializing the COP.

To load a previously saved configuration at initialization:

1. Create a Configuration object.
2. Call the Configuration object's fromXML method to read the previously saved configuration from an input stream.

(Alternatively, if the configuration has been stored in a format other than the Configurator's standard XML format, it may be necessary to read the saved configuration, parse it, and set the Configuration object's values directly by using its "set" methods - setModelName, setChoices, etc.)

3. Call restore, with the Configuration object as the config parameter. The COP will then be attached to the model identified by the model name, version, and compileID found in the config parameter, with the previously saved configuration data restored.

In all cases, you must also provide an appLocale and needHtmlEncoding parameter. appLocale specifies the language that should be used for user-readable text. needHtmlEncoding specifies whether attribute data returned by the COP should be in HTML-encoded format. When using the restore method you must also provide a policy. parameter. policy specifies how to handle the retrieval of the model to attach to before loading the configuration.

Policy options are:

Option	Description
1	Use the model version stored in the Configuration object. Fail if that version is not found.
2	Use the model version stored in Configuration object. Use the latest version if that version is not found.
3	Always use the latest model version.
4	If multiple versions exist, ask which to use. If only one version exists, automatically use it. If no versions exist, fail.

With option 4, if multiple versions exist, the restore method returns a status code indicating that there are multiple versions. The methods getModelVersions and getCompileVersions could then be used to determine available model versions. When the desired version is found, call the setModelVersion and/or setModelCompileVersion on the Configuration object. Once these initial calls are made, you can use the Configuration object in a subsequent call to restore with the policy parameter set to 1.

Additionally, you can specify the solve date for the COP to use when processing choices by calling the setSolveDate method. This method takes a java.util.Date object as a parameter. The COP constructor defaults the solve date to the current date. An example of overriding the default date is restoring a previously saved configuration. You may want to use the date that the configuration was saved as the solve date rather than the current date.

Example (without using a previously saved configuration): there are values for the model name, model version, and compileID, but if the compileID is empty (null or of zero length), we'll initialize without it.

```
ClientOperations cop = new ClientOperationsImpl();
String modelName, modelVersion, compileID;
{
  //set modelName, modelVersion, compileID here - not COP code
}
try
{
  if ((compileID == null) || (compileID.length() == 0))
  {
    cop.initialize(Locale.getDefault(), modelName, modelVersion, false);
    compileID = cop.getModelCompileVersion();
  }
}
else
```

```

{
    cop.initialize(Locale.getDefault(), modelName, modelVersion, compileID, false);
}

catch (Exception e)
{
    System.out.println("Error: Unable to initialize COP - " + e);
}

```

Releasing the COP

When you are done with your session, call `release`. This allows the system to dispose of cached resources associated with the session. After calling `release`, you will not be able to make `ClientOperations` calls without first calling `initialize` again.

Processing and Displaying a Page

Processing and displaying a page is one of the most common tasks to accomplish using the COP Java API. It consists of several subtasks.

Circumstances that require you to process and display a page are:

- Initial page display, before the user has made any choices. In this case, the “processing” is done to retrieve the model’s default choices and process a solution state for them.
- Subsequent page displays, after the user has made some choices and submitted a page.
- In some user interfaces, every time the user makes a choice the value is submitted to the Configurator Engine (this is called “auto-submit”). Auto-submit gives the user more immediate feedback. Obviously this has performance implications, since processing and redrawing are being done more frequently.

To process and display a page:

1. If you want delta-pricing information for one or more domain members and/or a total price for the configuration, call `setPricingData` if you are running the Configurator in stand-alone mode.

If you are running the Configurator integrated with Order Capture, setup for pricing data is handled within the Configurator schema setup pages.

See [Chapter 22, “Setting Up Integration,” Establishing Configuration Display and Pricing Options, page 284](#).

If you don’t want delta-pricing information, skip this step.

2. If the user has made domain member choices (which is usually the case, except for the initial page display), call `makeSelectedChoice` and `makeEliminatedChoice` to create one `DMChoice` object for each user choice or elimination.

The COP does not indicate which domain members in the UI have been selected or eliminated by the user; you must examine the UI controls and tell the COP by passing `Choice` objects to `processChoices` - see Step 4.

3. If the user has entered extern variable choices, call `makeExternVarChoice` to create one `EVChoice` object for each extern variable choice.
4. Call `processChoices`. *This step is mandatory.*

If there are user choices (domain member selections or eliminations, or extern variable choices), pass them to `processChoices` in a vector of `Choice` objects. This vector can contain both `DMChoice` objects and `EVChoice` objects, since both are subclasses of `Choice`.

If there are no selections, pass null to `processChoices`.

The Configurator Engine creates a solution state representing the choices and their implications—which domain members are user-selected, user-eliminated, computer-selected, computer-eliminated, and so on—and the COP generates any delta-pricing information you requested in Step 2. This information is cached by the COP until the next time you call `processChoices` (or `release`), and can be retrieved by calling `getControlData` (and other methods), as described in Step 5.

5. For each decision point for which you need display information, call `getControlData`.

This call will return a `ControlData` object representing that decision point. The `ControlData` object contains an array of `ControlItem` objects, representing the decision point's domain members. Also, the `ControlData` method iterator can be called to obtain iterators (objects of type `ItemIterator`), which are used to access the `ControlItem` objects.

When you call `getControlData`, you specify two parameters (*sort* and *filter*) that determine how you want domain members to be sorted and filtered. The `ControlData` object's iterator(s) will return the `ControlItem` objects in sorted order and filtered as specified. (You can further control the sort order by supplying a custom `Comparator` routine.)

See [Chapter 12, “Using the COP Java API,” Sorting and Filtering, page 174](#).

6. Using the `ControlData` and `ControlItem` classes, retrieve any display data you want: domain members (sorted and filtered as desired); their attribute values; and their states (such as user-selected and computer-eliminated).
7. If you wish to display the values of any extern variables, call the `getExternVar` method.
8. Render your page with the display information from Step 6 and Step 7.

In addition to the methods of the `ControlData` and `ControlItem` classes, you can use the `ClientOperations` methods `getTotalPrice`, `getViolations`, and `getNumericData`, and the `Configuration` method `getAttribute` for other information you might wish to display.

Getting a ControlData Object

In Step 5 of the preceding instructions for processing and displaying a page, you get one `ControlData` object for each decision point for which you need display information (or for whose domain members for which you need display information). You do this by calling the `getControlData` method.

This section describes that method in more detail. There are two forms of `getControlData`:

```
ControlData getControlData(String objectName, String attribute, boolean sort, Item⇒
Filter filter)
ControlData getControlData(String objectName, String[] attributes, boolean sort,⇒
ItemFilter filter)
```

Use the first form when you are only interested in obtaining values for a single attribute (one value for each domain member of the decision point). Use the second form when you are interested in the values of multiple attributes.

The call returns a `ControlData` object, representing a decision point. The `ControlData` object includes an array of `ControlItem` objects, representing all the domain members of that decision point. You can also obtain one or more iterators (objects of type `ItemIterator`) from the `ControlData` object, which are used to access the `ControlItem` objects.

In many cases, you will want the domain members in the UI to be sorted in a particular way. The sort order can be either a standard one implemented by the COP (based on the domain members' states and elimination levels), or a custom sort order based on a Comparator routine you supply.

See [Chapter 12, "Using the COP Java API," Sorting and Filtering, page 174.](#)

You may also want some domain members to be filtered out, so they don't appear in the UI at all. For example: filter out all eliminated domain members. Specify the sort and filter as parameters to the `getControlData` call. The `ControlData` object's iterator(s) will return the `ControlItem` objects (domain members) in the requested sort order, and filter out (skips over) any domain members as specified.

Note. Although you can access the `ControlItem` array directly using the standard array methods, you will almost never want to. The array is neither sorted nor filtered by the COP. Use the `ControlData` object's iterators; they will return the `ControlItem` objects in the correct sort order, and filter (skip over) any `ControlItem` outside the specified range.

The parameters to `getControlData` have the following meanings:

objectName	The name of the decision point.
attribute	In the first (single-attribute) form of the call, the name of the attribute whose values you wish to obtain. After making the <code>getControlData</code> call, you can get <code>ControlItem</code> objects from the resulting <code>ControlData</code> object, then get the attribute value for each <code>ControlItem</code> (domain member) by using the <code>ControlItem</code> method <code>getAttributes</code> (which, in this case, will return a vector with one element).
attributes	In the second (multiple-attributes) form of the call, an array containing the names of all the attributes whose values you wish to obtain. After making the <code>getControlData</code> call, you can get <code>ControlItem</code> objects from the resulting <code>ControlData</code> object, then get the attribute values for all the attributes you specified for each <code>ControlItem</code> (domain member) by using the <code>ControlItem</code> method <code>getAttributes</code> .
sort	If true, the <code>ControlItem</code> objects returned by an iterator (object of type <code>ItemIterator</code> , obtained by calling the <code>ControlData</code> object's iterator method) will be sorted. The sort order will either be a standard one implemented by the COP, or a custom sort order based on a <code>Comparator</code> object that you supply.

See [Chapter 12, "Using the COP Java API," Sorting and Filtering, page 174.](#)

The standard sort order implemented by the COP is based on the state and elimination level of the `ControlItem` objects (domain members). The domain members are sorted as follows:

1. Selected domain members in conflict.
2. Selected domain members not in conflict.
3. Selectable domain members.
4. Eliminated domain members—high elimination level.
5. Eliminated domain members—low elimination level.

If there is more than one domain member in a category, they will be sorted in their "default order." (This is the order in which they were originally supplied by the modeler or the data base.) In other words, within each category the

default order will be preserved. This applies to eliminated domain members only if their elimination level is equal.

If `sort` is true and you supply a `Comparator` object, your `Comparator` routine will be used to sort the domain members.

If `sort` is false, the COP will do no sorting of domain members. Any iterator obtained from this `ControlData` object's `iterator` method will return the domain members in their default order.

filter

Filters out certain `ControlItem` objects (domain members), based on their state and elimination level. The `ControlData` object's iterator simply skips over these `ControlItem` objects. For example: this can be used to filter out all eliminated items (so the user doesn't even see them), or all eliminated items with an elimination level below a certain number.

If `filter` is null, no filtering is done (except for "deleted" domain members).

See [Chapter 12, "Using the COP Java API," Handling Deleted Domain Members, page 176](#).

Given a decision point name (*dpName*) and attribute (*attribute*), get and display the domain members—sorted but not filtered.

```
ControlData ctrlData = null;
ControlItem ctrlItem = null;
boolean sortFlag = true;
ItemFilter filter = null;
ItemIterator ctrlItemIterator = null;
try
{
    ctrlData = cop.getControlData(dpName, attribute, sort⇒
Flag, filter);
}
catch (Exception e)
{
    System.out.println("Error: Unable to retrieve ctrlData ->
" + e);
    ctrlData = null;
}
if (ctrlData != null)
{
    ctrlItemIterator = ctrlData.iterator();
}
if (ctrlItemIterator != null)
{
    while (ctrlItemIterator.hasNext())
    {
        ctrlItem = (ControlItem) ctrlItemIterator.next();
        if (ctrlItem != null)
        {
            // you have ctrlItem - examine and display it here
        }
    }
}
```

```
}

```

It's important to note that the methods of `ControlData` and `ControlItem` objects return the most recent display information, which is based on the most recent calls to `processChoices`, `getControlData`, `setPricingData`, and so on, regardless of when the `ControlData` or `ControlItem` object was first created. They do not "remember" old values. So, for example, if you do the following:

1. Make some user choices.
2. Call `processChoices`.
3. Call `getControlData` to get a `ControlData` object for a decision point.
4. Make different user choices.
5. Call `processChoices` again.
6. Call `getFlags` on the `ControlData` object created in Step 3 to get the state of the decision point.

You would get the current state of the decision point, not the state it had as of Step 3. So there is no reason to save a `ControlData` or `ControlItem` object in order to hold old display information. It won't work. (You could, of course, call `getFlags` immediately after Step 3 and save the value itself.)

Specifying Delta-Pricing and Total-Pricing Requirements

The procedure to process and display a page specifies those domain members (if any) for which you need pricing-related information.

See [Chapter 12, "Using the COP Java API," Getting Display Information for a Domain Member, page 176](#).

There are two ways you can accomplish this:

- By calling the `setPricingData` method.

Use this method if you are running the Configurator in stand-alone mode. This section describes that method in more detail.

- By integrating Configurator with Order Capture.

Setup for pricing data is handled within the Configurator schema setup pages. When using this method of specifying pricing data, use the `ClientOperationsImpl` constructor that takes a `solutionId` as a parameter when creating the COP. The `solutionId` tells the COP which schema to reference for the appropriate pricing data.

Calling `setPricingData` describes a set of domain members for which you want pricing-related information. This information is cached by the Configurator (until you call `setPricingData` again, or release). The next time `getControlData` is called, the COP will be called to generate pricing-related information for those domain members. The delta-price value for a domain member can then be retrieved from the `ControlItem` representing the domain member using the `ControlItem` method `getDeltaPrice`. The total price for the configuration can also be retrieved using the `ClientOperations` method `getTotalPrice`.

There is one form of `setPricingData`:

```
void setPricingData(Map dpNamesAndAttributes)

```

This method takes a `Map` of decision point names and the corresponding price attribute for that decision point. Only the domain members in the specified decision points will be delta-priced. Also, only the specified decision points will be used when calculating the total price for the configuration.

Note. Configurator remembers only the most recent `setPricingData` call. Subsequent calls do not "accumulate" domain members for which to get delta-pricing data.

Getting Other Display Information

Some ClientOperations methods return information that you may wish to display, but that is not included in ControlData or ControllItem objects. These calls include:

- *getExternVar*—returns an extern variable (object of type ExternVar), based on its name. The ClientOperations method *getExternVarNames* returns a vector of the names of all the extern variables that are part of the model.
- *getTotalPrice*—returns the total price of the current configuration.
- *getViolations*—returns all violations associated with the configuration.
- *getNumericData*—returns a NumericData object. Some models contain variables that you may wish to display in the UI, such as “total grams of saturated fat” or “number of video card slots remaining,” that are distinct from domain member attributes, prices, and quantities. These variables are represented by NumericData objects. Given the name of such a variable, this call returns a NumericData object representing it. You can then use the NumericData class methods to get the variable’s type and value.

See [Chapter 12, “Using the COP Java API,” NumericData, page 183](#).

You may also wish to display one or more configuration attributes. To obtain them, call the ClientOperations method *getConfiguration* to get the current configuration, and then the Configuration method *getAttribute* for each attribute whose value you want to display.

Verifying a Configuration

When the user wishes to finalize the configuration, as in making a purchase, you can verify the configuration by calling the ClientOperations method *verifyConfiguration*. This method is similar to *processChoices*; it takes a vector of Choice objects (representing the configuration) as input, and calls the Configurator Engine to generate a solution state. The vector can contain both DMChoice objects and EVChoice objects. *verifyConfiguration* also returns a boolean value: true if the configuration is complete and consistent, false if otherwise. Complete means that every required (non-optional) decision point in the configuration has a selection. Consistent means the configuration has no violations.

Note. There is an alternate version of *verifyConfiguration* that takes a Configuration object as a parameter. With the exception of this additional parameter, it functions the same as the other version.

Configuration

This class represents a set of choices for the model, any configuration attributes, and the model’s name, version, and compileID.

The choices may include both DMChoice objects and EVChoice objects.

You can obtain the current configuration by calling the ClientOperations method *getConfiguration*. This configuration will include user, default, and computer choices.

The Configuration object can be used to write out the configuration to an external format (usually XML), or to read in a configuration that has been saved in that format. You can also directly set and get the information in the configuration: choices, configuration attributes, and the model’s name, version, and id.

You can also save and restore a configuration in other formats, but this requires a significant amount of custom code and is generally not worthwhile.

Methods

The Configuration class contains the following methods:

```

void convertElement(org.w3c.dom.Element elem)
void fromXML(InputStream is)
String getAttribute(String attribute)
Map getAttributeMap()
Vector getChoices()
String getModelCompileVersion()
String getModelName()
String getModelVersion()
String getTagName()
Vector getUserChoices()
String removeAttribute(String attribute)
void setAttribute(String attribute, String value)
void setChoices(Vector choices)
void setModelCompileVersion(String modelCompileVersion)
void setModelName(String modelName)
void setModelVersion(String modelVersion)
org.w3c.dom.DocumentFragment toXML(org.w3c.dom.Document doc)
Object clone()
DeltaConfig delta(Configuration newCfg)
boolean equals(Configuration newCfg)
Date getLastSavedDate()
Vector getNumericDatas()
boolean hasViolations()
void toXML(OutputStream out)
Date getSolveDate()

```

Saving and Restoring a Configuration

The COP can be used to save the current configuration in an external form, or to read a previously saved configuration and make it current. The COP uses a Configurator-defined XML format and standard Java input and output techniques for these operations. Provided the property `calico.na.db.compression` in the `Advisor.properties` file is set to `true`, it also compresses the data during the save and de-compresses it (if needed) during a restore.

To save the current configuration:

1. Call the ClientOperations method `getConfiguration` to obtain a Configuration object representing the current configuration.
2. Call that object's `toXML` method to write out an XML representation of the configuration to a specified output stream.

To restore a previously saved configuration (that is, make it current):

1. Create a Configuration object.
2. Call that object's `fromXML` method to read the previously saved configuration from a specified input stream.
3. Call the ClientOperations method `restore`, with the Configuration object as its `config` parameter.

Note. The Configuration methods toXML and fromXML assume the use of the Configurator's standard XML format for configurations. It is also possible to represent a configuration in some other format, defined either by the programmer or by some other (non-Configurator) standard. In this case you can not use toXML or fromXML. Instead, to save the configuration, you would examine each item in the configuration (including all the choices, configuration attributes, and model identification information) and write it out in the desired format. To restore a saved configuration, reverse the process: create a Configuration object, read in and parse the representation of the stored configuration, and set the values of all items in the Configuration object accordingly.

Obviously, using a non-standard format to represent a configuration entails much more work than using the Configurator's XML representation. It is generally not worthwhile.

ControlData

This class represents a decision point. You create a ControlData object for a decision point by calling the ClientOperations method getControlData. The ControlData object contains an array of ControlItem objects, representing all its domain members. It also enables you to obtain one or more iterators (objects of type ItemIterator) for accessing the ControlItem objects.

If you set the sort parameter of the getControlData call to true, then an iterator obtained from the ControlData object will return the ControlItem objects in a sort order determined by the software.

See [Chapter 12, "Using the COP Java API," Sorting and Filtering, page 174](#).

Iterators can also filter out (skip over) certain domain members, such as all eliminated domain members. Specify the filtering you want by using the filter parameter of getControlData.

See [Chapter 12, "Using the COP Java API," Sorting and Filtering, page 174](#).

Note. Although you can access the ControlItem array directly using the standard array methods, you will almost never want to. The array is neither sorted nor filtered by the COP. Use an iterator obtained from the ControlData object; the iterator will return the ControlItem objects sorted, and filter them as specified.

See [Chapter 12, "Using the COP Java API," Handling Deleted Domain Members, page 176](#).

Note. Each optional decision point has a special domain member with the reserved name "\$NADA", representing a choice of "none." This domain member is supplied by the Configurator Engine; it does not come from the modeler or an external source. It does not appear in either the ControlData object's ControlItem array or in the ControlItem objects returned by the ItemIterator. Instead, it can only be obtained by calling the ControlData method getControlItem, with a String parameter that has been set to \$NADA.

If, for a given optional decision point, you want to give your users the option of "none" (including displaying the option, displaying its state, and so on), you need to handle it specially, using the \$NADA domain member and the getControlItem call.

Note. The ControlData method isPublic indicates whether a decision point is public or private. However, the current version of the Configurator does not implement this distinction; all decision points are considered public.

Methods

The following methods are contained in the `ControlData` class:

```

Object[] getAllItems()
Choice[] getChoices()
String getClassID()
ControlItem getControlItem(String domainName)
long getFlags()
String getName()
double getQty()
Violation[] getViolations()
boolean hasDeletedItems()
boolean isMultiSelect()
boolean isOptional()
boolean qtySupported()
boolean isPublic()
ItemIterator iterator()
ItemIterator iterator(Comparator comp)
int getMinChoices()
int getMaxChoices()
String[] getAttributeNames()
String getAttributeValue(String name)
String[] getAttributeValues(String[] names)

```

Getting Display Information for a Decision Point and Its Domain Members

Most of the display information is associated with the decision point's domain members, rather than the decision point. To get at this information, call the `ControlData` method `iterator` to get an iterator, and use it to access the `ControlItem` objects. They will be sorted and filtered as specified by the software.

You can also call `getAllItems`, which returns the `ControlItem` array directly. But this array is not sorted or filtered by the COP, regardless of the `getControlData` parameters. It contains all of the decision point's domain members, in whatever order they were supplied by the Visual Modeler or an external source.

For each `ControlItem`, use the `ControlItem` methods to retrieve its display information.

See [Chapter 12, "Using the COP Java API," ControlItem, page 176](#).

There is some information that you may wish to display that applies to the decision point as a whole. Methods that return such information include:

- | | |
|----------------------|---|
| getQty | Returns the total of all the quantities for all the selected domain members of this decision point. Only some decision points, determined by the modeler, permit their domain members to have quantities. You can determine if a decision point permits quantities by using the <code>ControlData</code> method <code>qtySupported</code> . |
| getState | Returns the state of the decision point.

See Chapter 12, "Using the COP Java API," Getting the State of a Domain Member, page 177 . |
| getViolations | Returns all violations associated with this decision point. |

int getMinChoices	Returns the minimum selection quantity value for this decision point.
int getMaxChoices	Returns the maximum selection quantity value for this decision point.
String[] getAttributeNames()	Returns the names of the selection point attributes for this decision point.
String getAttributeValue(String name)	Returns the value of the specified attribute.
String[] getAttributeValues(String[] names)	Returns the specified attribute values.

Getting the State of a Decision Point

Based on the configuration and solution state at any given time, a decision point has one or more states, which can be obtained by calling the ControlData method getFlags. The state of a decision point depends completely on the state of its domain members.

See [Chapter 12, “Using the COP Java API,” Getting the State of a Domain Member, page 177.](#)

There are four decision point states:

selectable	If one or more of the domain members is selectable.
selected	If one or more of the domain members is selected.
conflicted	If one or more of the domain members is conflicted
undefined	Neither selectable, selected, nor conflicted.

Combinations are certainly possible. For example, a decision point can be simultaneously selected (because at least one of its domain members is selected) and selectable (because some other of its domain members are selectable).

All of a decision point’s domain members are used in determining the domain member’s state, not just the domain members that have not been filtered out by the ControlData object’s iterator(s).

The ControlData interface defines three different state bits, which can be used to test a selection point (decision point’s state flags:

Decision point state flags	Bits
SELECTABLE	0x01
SELECTED	0x02
CONFLICTED	0x04

Note. The particular values of the state bits are not important, and are only included for illustration. They might be changed in a later release. It is important that they are single, distinct bits. The bit values are not the same as the bit values for the similarly named field constants in the ControlItem interface.

The ControlData interface also defines:

Decision point state	Bits
UNDEFINED	0x00

You can use these field constants with the value returned by the `ControlData` method `getFlags` to determine the decision point's current state or states.

Sorting and Filtering

A `ControlData` object contains an array of `ControlItem` objects, representing its domain members. This array is not sorted or filtered by the COP. All the `ControlData` object's `ControlItem` objects (domain members) appear in the array. The `ControlItem` objects are arranged in the order that the corresponding domain members were loaded into the engine at model load time. For external domain members, this is the order specified by the SQL statement for the domain member query. For internal domain members, it is the order they were entered into the class by the modeler. This is called the domain member's default order.

However, the COP does permit you to sort and filter the `ControlItem` objects in certain ways. This is not implemented through the `ControlItem` array, but through objects of type `ItemIterator`, which can be obtained by calling the `ControlData` method `iterator`.

A `ControlData` object is created by a call to the `ClientOperation` method `getControlData`. If the `getControlData` parameter `sort` is set to `true`, then iterators (objects of type `ItemIterator`) obtained from the `ControlData` object's `iterator` method will return the `ControlItem` objects in a sorted order. This sort order will either be a standard one implemented by the COP, or a custom sort order based on a `Comparator` object that you supply, depending on which form of the `ControlData` method `iterator` you use.

You can also have multiple iterators at the same time, for simultaneous access to standard sorting, custom sorting (one or more), and multiple traverses of the `ControlData` object's domain members. But this will not usually be necessary.

Standard Sorting

The first form of the `ControlData` method `iterator` takes no parameters: `iterator()`. If you call this form, the resulting iterator returns the `ControlItem` objects sorted in a standard manner implemented by the COP. The sort order, one that many User Interfaces prefer, is as follows:

1. Selected domain members in conflict (that is, both selected and eliminated).
2. Selected domain members not in conflict.
3. Selectable domain members (neither selected nor eliminated).
4. Eliminated (but not conflicted) domain members—sorted from highest elimination value to lowest elimination value.

Within each category (for 1, 2, and 3), if the category has more than one domain member, its domain members are arranged in the same order in which they appear in the `ControlItem` array. That is, the original (default) order is maintained within each category.

Similarly, for category 4, multiple domain members with the same elimination value will be arranged in their original (default) order.

Every eliminated domain member has an elimination value. If it's not determined by constraints for which the modeler has defined elimination levels, it defaults to 1.

Custom Sorting

The second form of the `ControlData` method `iterator` takes a `Comparator` object as its parameter: `iterator(Comparator comp)`. If you call this form, the resulting iterator returns the `ControlItem` objects sorted in the order determined by your `Comparator` object. This enables you to create a completely customized sort order by writing your own `Comparator` object. Your `Comparator` will have access to all the methods of the `ControlItem` objects it is comparing, enabling you to sort on attribute values, states, and other domain member aspects.

Note. A `Comparator` is a Java object that is used to impose a sort order on a collection of objects by comparing pairs of objects and returning a result indicating which object is “greater” and which is “lesser?” (or that the two are to be considered “equal” in the sort order). In this case the objects are the `ControlItem` objects of a given `ControlData` object. `Comparators` implement the `java.util.Comparator` interface.

No Sorting

If you call the `ClientOperations` method `getControlData` with the sort parameter set to `false`, then the `ControlItem` objects of the resulting `ControlData` object will not be sorted. Any iterator returned by that `ControlData` object’s `iterator` method will return `ControlItem` objects in their original (default) order—exactly as they appear in the `ControlItem` array—with no sorting by the COP. This is true regardless of which form of the iterator method you call. However, filtering may still occur.

See [Chapter 12, “Using the COP Java API,” Sorting and Filtering, page 174](#).

Filtering

If the `getControlData` call has a non-null filter parameter, then iterators obtained from the resulting `ControlData` object will filter out (skip over) certain `ControlItem` objects, and only return the ones that are not filtered. This makes it easier for the UI to conceal from the user a certain class of domain members (commonly, all eliminated domain members, or eliminated domain members whose elimination value is below a certain number). The filtering depends on the filter parameter, as follows:

- If you call the first version of the `ItemFilter` method `setRange—setRange(boolean filterAllEliminatedItems)`—with the parameter set to `true`, the filter will filter out all eliminated (but not conflicted) items.
- If you call the second version of the `ItemFilter` method `setRange—setRange(int lower, int upper)`—then any eliminated (but not conflicted) domain members whose elimination values are outside the range from `lower` to `upper` (inclusive) will be filtered out.

Specifically, if you want to filter out all eliminated (but not conflicted) domain members with an elimination value less than some integer `N`, you would call `setRange(N + 1, ItemFilter.maxEliminationValue)`.

Every eliminated domain member has an elimination value. If it’s not determined by constraints for which the modeler has defined elimination levels, it defaults to 1.

- Regardless of the value of the filter parameter—even if it is `null`—the `ControlData` object’s iterators will always filter out “deleted” domain members—that is, an external domain member that has been removed from its database after the model was compiled. These deleted domain members are still present in `ControlData` object’s `ControlItem` array, so they can be retrieved if necessary.

See [Chapter 12, “Using the COP Java API,” Handling Deleted Domain Members, page 176](#).

If the `getControlData` range parameter is `null`, the `ControlItem` object’s iterators will only filter out deleted domain members. No other domain members will be filtered out.

Handling Deleted Domain Members

In systems that load domain members from external databases, it is sometimes possible to have “deleted” domain members—that is, an external domain member that has been removed from its database after the model was compiled. It might be necessary to inform the user that a domain member has been deleted, especially if it had been user-selected.

A `ControlData` object’s iterators automatically filter out (skips over) any deleted domain members, so they are not helpful in handling them. But the deleted domain members are still present in the `ControlData` object’s `ControlItem` array. Use the `ControlData` method `hasDeletedItem` to see if the decision point has a deleted domain member. If it does, you can use `getAllItems` to get the `ControlItem` array, then use the standard array operators to look at all the `ControlItem` objects in the `ControlItem` array. The `ControlItem` method `isValid` (false for deleted domain members, true otherwise) will tell you if a given domain member is deleted. If it is, you can then check the domain member’s state to determine if the user needs to be notified.

ControlItem

This class represents a domain member. It contains display information for that domain member.

Note. A `ControlItem` always comes from a `ControlData` object, which was originally obtained from a call to the `ClientOperations` method `getControlData`. So every `ControlItem` can be traced back to a particular `getControlData` call. The `attributes` parameter to the `getControlData` (or `attribute` parameter, for the other version of the call) determines which attribute values can be obtained from the `ControlItem`.

Methods

The `ControlItem` class contains the following methods:

```
Vector getAttributes()
String getAttributeValue(String attribute)
String getClassID()
double getDeltaPrice()
long getEliminationLevel()
long getFlags()
String getName()
double getQty()
Violation[] getViolations()
boolean hasEliminationLevel()
boolean isValid()
public double getMaxQty()
public double getMinQty()
void setDeltaPrice(double newDelta)
```

Getting Display Information for a Domain Member

You can use these `ControlItem` class methods to get display information for a domain member:

Method	Description
getAttributes	Returns a vector of attribute values for the domain member represented by this ControlItem. The attributes whose values will be returned are those that were included in the attributes input to the getControlData call that this ControlItem can be traced back to. The order of the values will be the same as the order of the attributes in that parameter. The UI could display these values in whatever format you decide.
getFlags	Returns the current state flags for the domain member. This tells you whether the domain member is user-selected, user-eliminated, computer-selected, computer-eliminated, and so on. The UI should mark the domain member appropriately. See Chapter 12, “Using the COP Java API,” Getting the State of a Decision Point, page 173.
getQty	Returns the quantity of this domain member. Only some decision points, determined by the modeler, permit their domain members to have quantities. You can determine if a decision point permits quantities by calling the ControlData method qtySupported on the associated ControlData object.
getDeltaPrice	Returns the delta price for this domain member. This information will only be available if the decision point this domain member belongs to was specified in the pricing definition setup.
getViolations	Returns an array of all the violations (if any) associated with this domain member.
isValid	Returns false for a deleted domain member, true otherwise. See Chapter 12, “Using the COP Java API,” Handling Deleted Domain Members, page 176.
hasEliminationLevel	Returns the elimination level for an eliminated domain member. Every eliminated domain member has an elimination value. If it’s not determined by constraints for which the modeler has defined elimination levels, it defaults to 1.
double getMinQty	Returns the minimum quantity value for this domain member.
double getMaxQty	Returns the maximum quantity value for this domain member.

Getting the State of a Domain Member

Based on the configuration and solution state at any given time, each domain member has one or more states, indicating whether the domain member is user-selected, computer-selected, user-eliminated, and so on. Multiple states are certainly possible. For example, a domain member can be both user-selected and computer-selected.

A domain member that is both selected and eliminated (for example, user-selected but computer-eliminated) is called conflicted. A domain member that is neither selected nor eliminated is called selectable.

The possible states for a domain member are: eliminated, selected, conflicted, selectable, computer-selected, default-selected, user-selected, computer-eliminated, default-eliminated, user-eliminated.

Note. The Configurator engine does not support “default-eliminated.”

The `ControllItem` interface defines five different state bits, which can be used to test a domain member's state flags:

Domain member state flag	Bits
ELIMINATED	0x01
SELECTED	0x02
COMPUTER	0x10
DEFAULT	0x20
USER	0x40

Note. The particular values of the state bits are not important, and are only included to make some of the examples clearer. It is important that they are single, distinct bits

These bits do not occur in isolation; you would not expect to find a domain member state with only one of these bits set. In fact, you would expect at least one from `ELIMINATED` and `SELECTED`, and at least one from `COMPUTER`, `DEFAULT`, and `USER`. These combinations define other states in the `ControllItem` interface:

Domain member state	Bits
<code>COMPUTER_ELIMINATED</code>	0x11
<code>DEFAULT_ELIMINATED</code>	0x21
<code>USER_ELIMINATED</code>	0x41
<code>COMPUTER_SELECTED</code>	0x12
<code>DEFAULT_SELECTED</code>	0x22
<code>USER_SELECTED</code>	0x42

The fields can be combined by the bitwise OR operation into a number that represents the domain member's state or states. This number is called the domain member's state flags (or just flags). You can use the `ControllItem` method `getFlags` to get the state flags for a domain member, and then use these field constants to determine the precise state or states of the domain member—and thus how to display it in your UI.

Note. If a domain member is conflicted, then it is both eliminated and selected. But it may be difficult or impossible to get more detailed information from the state flags—such as whether the domain member is computer-selected, user-selected, computer-eliminated, and so on. For example, a domain member that is user-selected and computer-eliminated has exactly the same state bits set as a domain member that is user-eliminated and computer-selected (namely, `USER`, `COMPUTER`, `SELECTED`, `ELIMINATED`). So there's no way to tell from the state flags if the domain member is, for example, computer-selected or computer-eliminated. The convention, in this case, is to say that the domain member is conflicted, selected, and eliminated—and not try to go into any more detail than that.

It's up to you to decide how to use the states for display purposes in your UI. For example, most User Interfaces will display a domain member that is simply user-selected (0x42) the same way they will display one that is user-selected and computer-selected (0x52)—with an icon indicating it is user-selected. In this case user selection is considered more important than computer selection, and the UI doesn't bother to indicate both states. However, other User Interfaces may be required to display more information depending on the model needs.

The mapping from state flags to states is not always obvious, and you need to be careful. This is particularly true when dealing with conflicted domain members. For example, suppose you want to see if a domain member is computer-selected. When you examine the COMPUTER bit and the SELECTED bit, they are both 1.

It is possible that the domain member is computer-eliminated and user-selected—a conflicted combination. This combination (0x53) has the COMPUTER and SELECTED bits set, but the domain member is not computer-selected. You need to check the SELECTED bit to rule out this and similar possibilities.

The following two examples illustrate the use of the state flags to determine the state or states of a domain member.

Example 1: Check to see if a domain member is computer-selected (a conflicted item is not considered computer-eliminated, regardless of the other state bits that are set).

```
long stateFlags = 0;
stateFlags = ctrlItem.getFlags();
if ((stateFlags & ControlItem.COMPUTER_SELECTED) == ControlItem.COMPUTER_SELECTED) =>
    && ((stateFlags & ControlItem.CONFLICTED) != ControlItem.CONFLICTED))
{
    // domain member is computer-selected
}
else
{
    //domain member is not computer-selected
}
```

Example 2: Classifying a domain member as conflicted, selected, selectable, or eliminated.

```
long stateFlags = 0;
stateFlags = ctrlItem.getFlags();
if (stateFlags & ControlItem.CONFLICTED) == ControlItem.CONFLICTED)
{
    // domain member is conflicted
}
else if ((stateFlags & ControlItem.SELECTED) == ControlItem.SELECTED)
{
    //domain member is selected (and not conflicted)
}
else if ((stateFlags & ControlItem.ELIMINATED) != ControlItem.ELIMINATED)
{
    //not selected, not eliminated - domain member is selectable
}
else
{
    //domain member is eliminated
}
```

Choice

This class is the superclass for DMChoice and EVChoice. Several important methods in the COP Java API (such as the ClientOperations methods processChoices and verifyConfiguration) either accept or return a vector of objects of type Choice. These vectors can include both objects of type DMChoice and objects of type EVChoice, since, by inheritance, both are objects of type Choice.

In this version of the Configurator, many Choice class methods are deprecated and reintroduced in DMChoice because they apply only to domain member choices and not extern variables. For this reason there are very few Choice methods that are actually called; instead, the appropriate DMChoice methods and EVChoice methods are called.

The Choice class contains the following methods:

```
String getDecisionPointName()
String getDomainMemberName() //Deprecated. Use DMChoice.getDomainMemberName
double getQuantity() //Deprecated. Use DMChoice.getQuantity
long getState() //Deprecated. Use DMChoice.getState
abstract String getTagName()
boolean isSelection() //Deprecated. Use DMChoice.isSelection
boolean isUserChoice()
void setDecisionPointName(String dpName)
void setState(long newState) //Deprecated. Use DMChoice.setState
Object clone()
boolean isDMChoice()
void toXML(StringBuffer buffer)
```

When called on an object of type EVChoice:

- The method getDecisionPointName() returns the name of the extern variable.
- The method getType() returns the type of the extern variable.
- The method setDecisionPointName() changes the name of the EVChoice object (that is, associates the EVChoice object with a new extern variable, specified by name).
- The method isUserChoice() returns *false* if the extern value was defaulted in the model, until the user overrides the default value, in which case it will return *true*.

DMChoice

This class is a subclass of Choice. Objects of type DMChoice represent a domain member choice, that is, a selection or elimination. You create DMChoice objects by calling the ClientOperations methods makeSelectedChoice and makeEliminatedChoice. These DMChoice objects (along with EVChoice objects) can be passed in a vector to the ClientOperations method processChoices. No DMChoice or Choice class methods are required for this.

The DMChoice objects created by makeSelectedChoice and makeEliminatedChoice represent, by default, user domain member choices. However, by using the DMChoice method setState, you can change them to represent computer or default domain member choices (or various combinations). This may be necessary when you are creating DMChoice objects to include in a Configuration object that represents a configuration containing user, computer, and default domain member choices.

Methods

The following methods are contained in the DMChoice class:

```
String getDomainMemberName()
double getQuantity()
long getState()
String getTagName()
boolean isSelection()
boolean isUserChoice()
void setState(long newState)
boolean equals(Object newChoice)
void toXML(OutputStream out)
```

Examining a DMChoice

Given an object of type DMChoice, you can determine what domain member choice it represents in the following manner:

- Call `getDecisionPointName` and `getDomainMemberName` to get the decision point and domain member for the choice.
- Call `isSelection` to determine whether the choice represents a user selection or a user elimination. `isSelection` returns *true* for selections, *false* for eliminations.
- If the DMChoice object represents a user selection, and the associated decision point supports quantities for its domain members, you can call `getQuantity` to get the quantity associated with the selection. To determine if the decision point supports quantities, use the method `qtySupported()` in the ControlDate class.
- Call `getState()` to get the current state flags of the domain member associated with the DMChoice object.

EVChoice

This class is a subclass of Choice. Objects of type EVChoice represent extern variables. The value of an extern variable is a collection of floating-point numbers, strings, dates, boolean values, or integers.

You create EVChoice objects by calling the ClientOperations method `makeExternVarChoice`. These EVChoice objects (along with DMChoice objects) can be passed in a vector to the ClientOperations method `processChoices`. No EVChoice or Choice methods are required for this.

Methods

The following methods are contained in the EVChoice class:

```
String getTagName()
int getValueCount()
Collection getValues()
void setValues(Collection values)
boolean equals(Object newChoice)
void toXML(OutputStream out)
Object clone()
boolean isUserChoice()
```

Examining an EVChoice

By using the Choice method `getDecisionPointName` on an `EVChoice` object, you can get the name of the associated extern variable. You can then use the `ClientOperations` method `getExternVar` to get an object of class `ExternVar` representing the extern variable itself. Alternatively, you can bypass the `ExternVar` object and get the value of the extern variable associated with the `EVChoice` by using the `EVChoice` method `getValues`.

ItemFilter

The `ClientOperations` method `getControlData` has an `ItemFilter` parameter that enables the method to filter out certain domain members.

The most common use of an `ItemFilter` parameter is to filter out all eliminated domain members, or all eliminated domain members whose elimination level is below a certain threshold. Depending on the model and the User Interface, these are domain members that the user may not be interested in.

The `ItemFilter` parameter to `getControlData` causes the `ItemIterator` (belonging to the `ControlData` object created by the `getControlData` call) to filter out (skip over) the specified domain members. This makes it easy for the UI not to display those domain members. Note that the filtering is done by the `ItemIterator`; the `ControlData` object's `ControlItem` array still contains all the `ControlItem` objects (domain members) of the `ControlData` object (decision point).

Methods

The following methods are contained in the `ItemFilter` class:

```
void setRange(boolean filterAllEliminatedItems)
void setRange(int lower, int upper)
```

Filtering Out Domain Members

There are two versions of the `ItemFilter` method `setRange`. One takes a boolean parameter, `filterAllEliminatedItems`. The other takes two integers, `lower` and `upper`. Before using a parameter of type `ItemFilter` do one of the following:

- Call `setRange(true)` on the parameter to filter out all eliminated (but not conflicted) domain members; or
- Call `setRange(lower, upper)` on the parameter to filter out those eliminated (but not conflicted) domain members whose elimination values are outside the range from `lower` to `upper` (inclusive).

Specifically, if you want to filter out all eliminated (but not conflicted) domain members with an elimination value less than some integer `N`, you would call `setRange(N + 1, maxEliminationValue)`.

Every eliminated domain member has an elimination value. If it's not determined by constraints for which the modeler has defined elimination levels, it defaults to 1.

Note. Regardless of the value of the filter parameter—even if it is null—the `ControlData` object's iterators will always filter out “deleted” domain members—that is, an external domain member that has been removed from its database after the model was compiled. These discarded domain members are still present in `ControlData` object's `ControlItem` array, so they can be retrieved if necessary.

See [Chapter 12, “Using the COP Java API,” Handling Deleted Domain Members, page 176](#).

ItemIterator

The `ControlData` method `iterator` returns an object of the `ItemIterator` class. The `ItemIterator` object, called an iterator, is used to access the `ControlItem` objects (domain members) of the `ControlData` object (decision point). The iterator returns the domain members in a specified sort order, which may either be a standard sort order implemented by the COP or a custom sort order determined by a `Comparator` object you supply. The iterator also filters out (skips over) any domain members specified in the filter parameter to that `getControlData` call, as well as any deleted domain members.

It is possible to have more than one iterator for the same `ControlData` object at the same time. However, this is not usually necessary.

```
ItemIterator implements the Java interface java.util.Iterator.
```

The following methods are contained in the `ItemIterator` class:

```
boolean hasNext()  
Object next()
```

ExternVar

This class represents an extern variable, and can be used to get the name, type, and value of the extern variable. The value of an extern variable is a collection of floating-point numbers, strings, dates, boolean values, or integers.

The `ClientOperations` method `getExternVarNames` returns a vector of the names of all the extern variables in a model. The `ClientOperations` method `getExternVar` can then be used to retrieve the extern variable itself.

The value of an extern variable can be used as a choice in a configuration.

See [Chapter 12, “Using the COP Java API,” Handling Deleted Domain Members, page 176](#).

The `ExternVar` class methods are:

```
String getName()  
Collection getValues()  
String getType()  
Violation[] getViolations()  
boolean isDefault()
```

NumericData

This class represents a variable in the model. Some models contain variables that you might wish to display in the UI, such as “total grams of saturated fat” or “number of video card slots remaining.” These are distinct from domain member attributes, prices, and quantities. Obtain a `NumericData` object representing a model variable by using the `ClientOperations` method `getNumericData`. Then you can use the methods in this class to get the type and value of the variable.

Numeric data values are generated by the Configurator Engine, based on formulas specified in the model. From the point of view of the COP, they are read-only.

The NumericData class:

```
String getName()  
String getValue()  
String getType()  
boolean equals(Object obj)
```

Violation

This class returns information on a violation. Retrieve Violation objects associated with a configuration, a given decision point, or a given domain member by using the getViolations method of the ClientOperations, ControlData, or ControlItem classes, respectively.

The Violation class has one method, getExplanation, which returns a user-readable explanation for the violation.

The Violation method contains this method:

```
String getExplanation()
```

CHAPTER 13

Understanding the Configurator XML Interface

This chapter provides an overview of the Advanced Configurator XML interface and discusses:

- Request-response.
- Elements and attributes.
- Retrieving model information.
- Updating a configuration.

Request-Response

The COPXMLServlet translates COP XML to COP Java calls, and COP Java returns to COP XML. The servlet is installed with Advanced Configurator. If you installed Advanced Configurator in the default directory, the COPXMLServlet is in this file:

```
.\bea\weblogic81\config\Calico\applications\Calico\WEB-INF\lib\advisor.jar
```

You post a request to, and receive a response from, the servlet at COPXML on the computer where you installed the Configurator; for example, to post directly to the application server, use an URL like this:

```
http://MyComputer:7777/copxml
```

The COPXMLServlet and COP XML do not mimic the COP Java API. Instead, in one round trip, the COPXMLServlet:

- Bundles all of the requests included in the COP XML HTTP POST request.
- Makes all of the necessary COP Java calls required to process the requests.
- Processes everything that is returned by the COP, bundles it, and returns it in a single COP XML HTTP POST response.

Elements and Attributes

The COP XML request may include these elements:

```
CONFIGURATION  
DECISION_POINTS  
CHOICES  
CH  
EVCH  
CONTROL_DATA
```

```
ATTR
DP
ATTR
NUMERIC_VALUES
EXTERN_VARS
EV
VIOLATIONS
```

The Configurator XML interface lets you retrieve information about a model, including:

- Model name, version, and compile version, using attributes of the CONFIGURATION element.
- Error messages for an invalid element name, attribute name, or attribute value; omitting an attribute; or an abnormal processing error.
- A set of decision points, using the ALL attribute of the DECISION_POINTS element.

The CONFIGURATION element is required. Other elements are optional. Attributes of elements further specify the request. The COP XML response can include other elements and attributes, but is similar to the request.

The COP XML request-response pair can:

- Retrieve information about a model.
- Update a configuration (interactive mode).
- Retrieve information about a configuration (interactive mode).
- Create, update, save, or retrieve a configuration (batch mode).
- Change the order status of a configuration.

Retrieving Model Information

The Configurator XML interface lets you retrieve information about a model, including:

- Model name, version, and compile version.
- Error messages for an invalid request, or an abnormal processing error.
- A list of decision points.

See Also

[Chapter 14, “Retrieving Model Information,” page 189](#)

Updating a Configuration Interactively

The Configurator XML interface lets you update a configuration interactively by making choices or entering values for an external variable, that is, by:

- Selecting a domain member for a single-select decision point.

- Selecting one or more domain members for a multi-select decision point.
- Entering one or more values for an external variable.

See Also

[Chapter 15, “Updating a Configuration,” page 195](#)

Retrieving Configuration Information

The Configurator XML interface lets you retrieve this information about a configuration:

- Total price.
- Choices.
- Domain member data, including:
 - Domain members for all, or selected decision points.
 - Constraint violation explanations for decision points.
 - Value of domain member attributes.
 - Delta prices for domain members.
 - Changes that occurred between two versions of the configuration.
 - Class name for domain members.
 - State, and quantity of domain members.
- Whether a decision point is multi-select.
- Constraint violation explanations for the configuration.
- Numeric values.

The Configurator XML interface also lets you:

- Sort domain members that are returned by the COP.
- Filter domain members that are returned by the COP.

See Also

[Chapter 16, “Retrieving Configuration Information,” page 201](#)

Copying a Configuration

The Configurator XML interface lets you copy a configuration.

See [Chapter 18, “Copying a Configuration,” page 263](#).

Using Batch Configuration Mode

The Configurator XML interface lets you create, update, save and/or retrieve configurations in batch mode. See [Chapter 19, “Using Batch Configuration Mode,” page 265](#).

Changing the Order Status of a Configuration

The Configurator XML interface lets you change the order status of a configuration. See [Chapter 20, “Changing the Order Status of a Configuration,” page 269](#).

COP.dtd

COP.dtd defines which XML elements and attributes the Configurator XML interface can use to send requests to, and return responses from, the COP Java interface. COP.dtd also defines which elements and attributes the COP may use to save and restore configurations.

Studying the COP.dtd may help you better understand the COP XML that is used by the Configurator XML interface. An annotated version is located in `\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\Web-inf\dtd`.

Element-Attribute Trees

COP XML is defined by the COP.dtd. This means that some of its elements and attributes are not used by the Configurator XML interface, but are used by the COP to save and restore configurations.

COP XML may be viewed as a tree of elements, with some elements having attributes. Studying these element-attribute trees—especially trees that have only those elements and attributes that may be used in a request or response—may help you better understand the COP XML that is used by the Configurator XML interface.

See Also

[Appendix H, “Element-Attribute Trees,” page 509](#)

CHAPTER 14

Retrieving Model Information

The Configurator XML interface lets you retrieve information about a model, including:

- Model name, version, and compile version, using attributes of the CONFIGURATION element.
- Error messages for an invalid element name, attribute name, or attribute value; omitting an attribute; or an abnormal processing error.
- A set of decision points, using the ALL attribute of the DECISION_POINTS element.

Elements and Attributes

The COP XML request may include these elements and attributes to retrieve information about a model.

```
CONFIGURATION MODEL_ID LOCALE MODEL_VERSION COMPILE_VERSION  
DECISION_POINTS ALL
```

The COP XML response may include these elements and attributes to return information about a model.

```
CONFIGURATION MODEL_ID LOCALE MODEL_VERSION COMPILE_VERSION TOTAL_PRICE  
DECISION_POINTS  
DP NM  
ERROR
```

Version and Compile Version

The CONFIGURATION element, and its MODEL_ID, MODEL_VERSION, and COMPILE_VERSION attributes let you retrieve the model version and model compile version.

Any COP XML request must include at least the top level CONFIGURATION element, with at least the MODEL_ID and MODEL_VERSION attributes. The value for the MODEL_ID attribute must be the name of a valid model. The value for the MODEL_VERSION attribute may be a valid version number, valid partial version number, or an empty string.

Latest Version and Compile Version

The CONFIGURATION element, without a value for its MODEL_VERSION attribute, lets you retrieve the latest version and compile version for the model.

Including in the REQUEST

The CONFIGURATION element with:

- A valid value (model name) for the MODEL_ID attribute, and
- An empty string as the value for the MODEL_VERSION attribute.

For example:

REQUEST

```
<CONFIGURATION MODEL_ID="Sample"⇒
MODEL_VERSION="" />
```

Causes the COP to return in the RESPONSE

The CONFIGURATION element with values of:

- The given model name for the MODEL_ID attribute.
- The latest model version for the MODEL_VERSION attribute.
- The latest compile version for the COMPILE_VERSION attribute.

RESPONSE

```
<CONFIGURATION MODEL_ID="Sample" MODEL_⇒
VERSION=⇒
"1-0-0" COMPILE_VERSION="2000103⇒0-102408-⇒
606">
</CONFIGURATION>
```

Latest Compile Version

The CONFIGURATION element, with a valid value (full or partial version) for its MODEL_VERSION attribute lets you retrieve the latest version and compile version for the given version.

Including in the REQUEST

The CONFIGURATION element with valid values for:

- The MODEL_ID attribute.
- The MODEL_VERSION attribute.

For example:

REQUEST

```
<CONFIGURATION MODEL_ID="Sample"⇒
MODEL_VERSION="1" />
```

```
<CONFIGURATION MODEL_ID="Audio"⇒
MODEL_VERSION="3-1" />
```

Causes the COP to return in the RESPONSE

The CONFIGURATION element with values of:

- The given model name for the MODEL_ID attribute.
- The full, latest, given model version for the MODEL_VERSION attribute.
- The latest compile version for the COMPILE_VERSION attribute.

RESPONSE

```
<CONFIGURATION MODEL_ID="Sample" MODEL_⇒
VERSION="1-0-0"
COMPILE_VERSION="2000103⇒0-⇒
102408-606">
```

```
<CONFIGURATION MODEL_ID="Audio" MODEL_⇒
VERSION="3-1-0"
COMPILE_VERSION="2000110⇒5-⇒
115320-323">
</CONFIGURATION>
```

Note. The response for any valid request always includes the full model version and compile version. You may want to include the version and compile version from the first response in all subsequent requests.

Error Messages

An abnormal processing error or a request with invalid or omitted information causes the COP XML to return an error message.

Including in the REQUEST

One of the following:

- An invalid element name.
- An invalid attribute name.
- An invalid value for a valid attribute name.

OR omitting a required attribute

For example:

Causes the COP XML to return in the RESPONSE

The ERROR element as a child element of the CONFIGURATION element. The ERROR element's content is a textual description of the error.

REQUEST

(An invalid element name)

```
<CONFIGURATION/>
```

(An invalid attribute name)

```
<CONFIGURATION MODEL_ID=" "/>
```

(An invalid value for an attribute)

```
<CONFIGURATION MODEL_ID="Sample_2"=>
/>
```

RESPONSE

```
<CONFIGURATION>
<ERROR>com.calicotech.xml.ffInvalidChild=>
Exception: Child CONFIGURATION found where=>
CONFIGURATION required</ERROR>
</CONFIGURATION>
```

```
<CONFIGURATION>
<ERROR>calico.configurator.exceptions.COPExce=>
=>
=>
=>
=>
=>
ption: Problem TestModel:0-0-0 not found.<=>
/ERROR>
</CONFIGURATION>
```

```
<CONFIGURATION>
<ERROR>calico.configurator.exceptions.COPExce=>
=>
=>
=>
=>
=>
ption: Problem Sample_2:0-0-0 not found.<=>
/ERROR>
</CONFIGURATION>
```

REQUEST

(An invalid value for an attribute)

```
<CONFIGURATION MODEL_ID="Sample"
=>
MODEL_VERSION="1-1"/>
```

(Omitting the required MODEL_VERSION attribute)

```
<CONFIGURATION MODEL_ID="Sample"/>
```

RESPONSE

```
<CONFIGURATION>
<ERROR>calico.configurator.exceptions.COPExce=>
=>
=>
=>
=>
=>
ption: Version not found for Sample:1-1.<=>
/ERROR>
</CONFIGURATION>
```

```
<CONFIGURATION>
<ERROR>calico.configurator.exceptions.COPExce=>
=>
=>
=>
=>
=>
ption: Version not found for Sample:0-0-0.<=>
/ERROR>
</CONFIGURATION>
```

Decision Points

The DECISION_POINTS element and its ALL attribute let you retrieve two different sets of decision points for the model.

All Decision Points

The DECISION_POINTS element, with its ALL attribute set to "true" lets you retrieve every decision point for the model.

By default, including in the REQUEST the DECISION_POINTS element also includes the element's ALL attribute with its value set to true.

REQUEST

```

<CONFIGURATION MODEL_ID=
"Sample" MODEL_VERSION=""><DECI⇒
⇒
⇒
⇒
⇒
S⇒
ION_POINTS/>
</CONFIGURATION>

<CONFIGURATION MODEL_ID=
"Sample" MODEL_VERSION=""><DECI⇒
⇒
⇒
⇒
S⇒
ION_POINTS ALL="true"/>
</CONFIGURATION>

```

RESPONSE

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-⇒
⇒
⇒
⇒
⇒
0-0" COMPILE_VERSION="20001030-102408-606">
<DECISION_POINTS ALL="true">
  <DP NM="BaseSelection"/>
  <DP NM="HardDrivesSelection"/>
  <DP NM="DVDorCDRomSelection"/>
  <DP NM="OperatingSystemSelection"/>
  <DP NM="DeviceControllersSelection"/>
  <DP NM="MonitorsSelection"/>
  <DP NM="WattsDisplaySelection"/>
  <DP NM="ChassisSelection"/>
  <DP NM="PowerCordSelection"/>
  <DP NM="WarrantySelection"/>
  <DP NM="AdditionalSoftwareSelection"/>
  <DP NM="TravelSelection"/>
  <DP NM="PrimaryPurposeSelection"/>
  <DP NM="UserTypeSelection"/>
  <DP NM="SpeakersSelection"/>
  <DP NM="PrintersSelection"/>
  <DP NM="ScannersSelection"/>
  <DP NM="TapeBackDriveSelection"/>
  <DP NM="ZipDriveSelection"/>
  <DP NM="CDRWSelection"/>
  <DP NM="ModemSelection"/>
  <DP NM="SoundSelection"/>
  <DP NM="NetworkSelection"/>
  <DP NM="MemorySelection"/>
  <DP NM="VideoSelection"/>
</DECISION_POINTS>
</CONFIGURATION>

```

Note. Either request in the example causes the COP to return in the RESPONSE the DECISION_POINTS element with its ALL attribute set to “true”, and, as child elements, all of the DP elements for the model. The name of each decision point is the value of the NM attribute for each DP element.

Public Decision Points

The DECISION_POINTS element, with its ALL attribute set to “false” lets you retrieve every public decision point for the model.

Including in the REQUEST

The DECISION_POINTS element, with its ALL attribute set to "false."

- The MODEL_ID attribute.
- The MODEL_VERSION attribute.

Causes the COP to return in the RESPONSE

The CONFIGURATION element with values of:

- The DECISION_POINTS element with its ALL attribute set to "false".
- As child elements, only the public DP elements for the model.

For example:

REQUEST

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="" >
<DECISION_POINTS ALL="false"/>
</CONFIGURATION>
```

RESPONSE

Causes the COP to return only the public decision points for the Sample model.

CHAPTER 15

Updating a Configuration

This chapter discusses

- Updating a configuration.
- Elements and attributes
- Choices
- Choices and Response

See Also

[Chapter 13, “Understanding the Configurator XML Interface,” Retrieving Model Information, page 186](#)

[Chapter 13, “Understanding the Configurator XML Interface,” Retrieving Configuration Information, page 187](#)

Updating a Configuration

The Configurator XML interface lets you update a configuration by making choices—that is, by selecting a domain member for a single-select decision point, or one or more domain members for a multi-select decision point, or entering one or more values for an external variable.

To select a domain member using COP XML, the request must include:

- A CHOICES element.
- The child element CH for each domain member that you want to select.
- These attributes of the child element CH:
 - DP—The name of the decision point for the domain member you want.
 - DM—The name of the domain member you want.
 - BY—Indicates that the choice is made by the user (U).
 - QTY—The number of copies of the domain member you want.

To enter values for external variables using COP XML, the request must include:

- A CHOICES element.
- The child element EVCH for each external variable that you want to enter a value(s) for.
- These attributes of the child element EVCH:
 - DP—the name of the decision point for the external variable you want.

- VAL—the value you want to set on the external variable.

Note. You must enter a separate VAL attribute for each of the values you want to set on the external variable.

Note. The Configurator is stateless. Each request must include all user choices.

Elements and Attributes

The COP XML request may include these elements and attributes to update a configuration.

```
CONFIGURATION MODEL_ID LOCALE MODEL_VERSION COMPILE_VERSION TOTAL_PRICE
CHOICES RET
  CH DP DM BY QTY
  EVCH DP VAL
```

The COP XML response may include these elements and attributes to return information about a configuration.

```
CONFIGURATION MODEL_ID LOCALE MODEL_VERSION COMPILE_VERSION TOTAL_PRICE
CHOICES
  CH DP DM BY ST SL EL QTY
  EVCH DP VAL TY
```

Choices

The RET attribute of the CHOICES element lets you control whether the COP returns a response when you choose one or more domain members.

By default, including in the REQUEST the CHOICES element also includes the element's RET attribute with its value set to "false".

As mentioned above, a CHOICES request to update the configuration must also include, as a child element of the CHOICES element, EITHER the CH element with valid values for:

- The DP attribute
- The DM attribute
- The BY attribute
- The QTY attribute

OR the EVCH element with valid values for:

- The DP attribute
- The VAL attribute(s)

For example, these requests are, in effect, the same:

REQUEST 1 (RET not set):

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
```

```

    <CH DP="PrintersSelection" DM="HP Laserjet" BY="U"
QTY="1"/>
  </CHOICES>
</CONFIGURATION>

```

REQUEST 2 (RET set to an empty string):

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES RET="">
    <CH DP="PrintersSelection" DM="HP Laserjet" BY="U"
QTY="1"/>
  </CHOICES>
</CONFIGURATION>

```

REQUEST 3 (RET set to “false”)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES RET="false">
    <CH DP="PrintersSelection" DM="HP Laserjet" BY="U"
QTY="1"/>
  </CHOICES>
</CONFIGURATION>

```

Each of these requests—REQUEST 1, 2, and 3—updates the configuration by making a user choice of one copy of the HP Laserjet for the printer’s decision point. None of the requests cause the COP to return a RESPONSE.

Note. The Configurator is stateless. Each request must include all user choices.

Choices and Response

Including in the REQUEST the CHOICES element with the RET attribute set to “true” causes the COP to return in the RESPONSE:

- Every non-null attribute-value pair for every choice (pick) of a domain member in the configuration. The attributes of a domain member (DM) that may be returned are:
 - DP
 - DM
 - BY
 - QTY
 - ST
 - SL
 - EL
 - TY
- Every non-null, attribute-value pair for every external variable in the configuration.

The attributes of an external variable that may be returned are:

- DP
- VAL
- TY

For example:

REQUEST

(Without the CH child element)

```
<CONFIGURATION MODEL_ID=
"Sample" MODEL_ =>VERSION=" "><CHOICES=>
RET="true"/>
</CONFIGURATION>
```

(With the CH element PrintersSelection)

```
<CONFIGURATION MODEL_ID=
"Sample" MODEL_ =>VERSION=" "><CHOICES=>
RET="true">
  <CH DP="PrintersSelection" DM="HP=>
Laserjet" BY="U" QTY="1"/>
</CHOICES>
</CONFIGURATION>
```

(With the CH element BaseSelection)

```
<CONFIGURATION MODEL_ID=
"Sample" MODEL_ =>VERSION=" "><CHOICES=>
RET="true">
  <CH DP="BaseSelection" BY="U" QTY=>
"1" DM="Prima Base Celeron=>
433Mhz MiniTower"/>
</CHOICES>
</CONFIGURATION>
```

RESPONSE

```
<CONFIGURATION MODEL_ID="Sample" MODEL_ =>
VERSION="1-0-0" COMPILE_VERSION="20001128=>
143111-404">
<CHOICES>
  <CH DP="HardDrivesSelection" DM="4.3 GB=>
BATRA-33" BY="D" ST="34"/>
  <CH DP="ChassisSelection" DM="Mini Tower"=>
BY="C" ST="18"/>
</CHOICES>
</CONFIGURATION>
```

```
<CONFIGURATION MODEL_ID="Sample" MODEL_ =>
VERSION="1-0-0" COMPILE_VERSION="20001030=>
102408-606">
<CHOICES>
  <CH DP="PrintersSelection" DM="HP Laserjet"=>
=>
=>
=>
=>
BY="U" ST="66"/>
</CHOICES>
</CONFIGURATION>
```

```
<CONFIGURATION MODEL_ID="Sample" MODEL_ =>
VERSION="1-0-0" COMPILE_VERSION="20001030=>
102408-606">
<CHOICES>
  <CH DP="BaseSelection" DM="Prima Base=>
Celeron 433Mhz MiniTower" BY="U" ST="66"/>
  <CH DP="HardDrivesSelection" DM="4.3 GB=>
BATRA-33" BY="D" ST="34"/>
  <CH DP="MonitorsSelection" DM="15 Inch=>
Monitor" BY="D" ST="34"/>
  <CH DP="ChassisSelection" DM="Mini Tower"=>
BY="C" ST="18"/>
  <CH DP="PowerCordSelection" DM="10 Foot PC=>
Power Cord" BY="C" ST="18"/>
  <CH DP="AdditionalSoftwareSelection" DM=>
"ISP Direct Connect" BY="C" ST="18"/>
  <CH DP="TravelSelection" DM="No" BY="C" ST=>
=>
=>
=>
=>
"18"/>
  <CH DP="ModemSelection" DM="Internal 56K"=>
BY="D" ST="34"/>
</CHOICES>
</CONFIGURATION>
```

Note. The Configurator is stateless. Each request must include all user choices.

CHAPTER 16

Retrieving Configuration Information

This chapter discusses how to retrieve configuration information using COP XML:

- Elements and attributes
- Total Price
- Domain member data
- Multi-select decision points
- Global explanations
- Numeric values

Understanding Configuration Information

The Configurator XML interface lets you retrieve information about a configuration, including:

- Total price, using the TOTAL_PRICE attribute of the CONFIGURATION element.
- Choices—using the CHOICES element, the child elements CH and EVCH, and attributes of both elements.
- Domain member data, including:
 - Every domain member for every decision point in the configuration, using the CONTROL_DATA element.
 - Every domain member for specific decision points in the configuration, using the CONTROL_DATA element, the DP element, and the NM attribute of the DP element.
 - Constraint violation explanations for decision points, using the EXPLANATIONS attribute of the CONTROL_DATA element.
 - Values for domain member attributes, using the ATTR element and its NM attribute.
 - Delta prices for domain members, using the DPR attribute of the DP element.
 - Class name for domain members, using the CL attribute of the DP element.
 - State and quantity of domain members, using the DP element.
- Whether a decision point is multi-select, using the MS element.
- Global explanations—that is, constraint violation explanations for the configuration as a whole—using the VIOLATIONS element and its EXPLANATIONS attribute.
- Numeric values, using the NUMERIC_VALUES element, the child element NUM, and the NM attribute of the NUM element.

The Configurator XML interface also lets you:

- Sort domain members that are returned by the COP by state using the DMSORT_ST attribute of the CONTROL_DATA element.
- Filter domain members that are returned by the COP by elimination level using the FILTER_EL, FILTER_LO, and FILTER_HI attributes of the CONTROL_DATA element.

Note. The Configurator is stateless, so the examples in this chapter include user choices (picks) to create configurations that provide appropriate control data.

See Also

[Chapter 15, “Updating a Configuration,” page 195](#)

Elements and Attributes

The COP XML request may include these elements and attributes to retrieve information about a configuration.

```

CONFIGURATION MODEL_ID LOCALE MODEL_VERSION COMPILE_VERSION TOTAL_PRICE
CHOICES RET
  CH DP DM BY QTY
  EVCH DP VAL
CONTROL_DATA DMSORT_ST FILTER_EL_LO FILTER_EL_HI
  FILTER_EL EXPLANATIONS
  ATTR NM
  DP NM CL DPR
  ATTR NM
NUMERIC_VALUES
EXTERN_VARS
  EV NM TY
  VAL
VIOLATIONS EXPLANATIONS

```

The COP XML response may include these elements and attributes to return information about a configuration. Attributes are in small italic print.

```

CONFIGURATION MODEL_ID LOCALE MODEL_VERSION COMPILE_VERSION TOTAL_PRICE
DECISION_POINTS
  DP NM
CHOICES
  CH DP DM BY ST SL EL QTY TY
  EVCH DP VAL TY
CONTROL_DATA
  DP NM CL MS
  DM NM CL ST QTY SL EL PR
  ATTR NM
  EXPLANATION
NUMERIC_VALUES
  NUM NM VL
EXTERN_VARS
  EV NM

```

VIOLATIONS EXPLANATIONS
EXPLANATION

Total Price

The CONFIGURATION element and its TOTAL_PRICE attribute let you retrieve the configuration's total price.

Including in the REQUEST the CONFIGURATION element with an empty string value for its TOTAL_PRICE attribute causes the COP to return in the RESPONSE the value for the TOTAL_PRICE attribute.

For example:

REQUEST

```
<CONFIGURATION MODEL_ID="Sample" TOTAL_PRICE="" />
```

RESPONSE

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILE_VERSION="20000426=>  
113936-314" TOTAL_PRICE="964.95" />
```

Choices

The CHOICES element, its children elements CH and EVCH, and attributes of these elements let you retrieve choices for the configuration.

Including the CHOICES element in the request updates the configuration by setting choices. Setting the RET attribute of the CHOICES element to "true" in the request causes the COP to return all choices in the configuration.

More accurately, including in the REQUEST the CHOICES element with the RET attribute set to "true" causes the COP to return in the RESPONSE:

- Every non-null attribute-value pair for every choice (pick) of a domain member in the configuration. The attributes of a domain member (DM) that may be returned are:
 - DP
 - DM
 - BY
 - QTY
 - ST
 - SL
 - EL
- Every non-null, attribute-value pair for every external variable in the configuration.

The attributes of an external variable that may be returned are:

- DP

- VAL
- TY

For example:

REQUEST

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES RET="true">
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="MonitorsSelection" BY="U" QTY="1" DM="15 Inch Monitor"/>
  </CHOICES>
</CONFIGURATION>
```

RESPONSE

```
CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILE_VERSION="20001030=>
102408-606">
  <CHOICES>
    <CH DP="BaseSelection" DM="Prima Base Celeron 433Mhz MiniTower" BY="U" ST="66"/>
    <CH DP="HardDrivesSelection" DM="4.3 GB BATRA-33" BY="D" ST="34"/>
    <CH DP="MonitorsSelection" DM="15 Inch Monitor" BY="U" ST="66"/>
    <CH DP="ChassisSelection" DM="Mini Tower" BY="C" ST="18"/>
    <CH DP="PowerCordSelection" DM="10 Foot PC Power Cord" BY="C" ST="18"/>
    <CH DP="AdditionalSoftwareSelection" DM="ISP Direct Connect" BY="C" ST="18"/>
    <CH DP="TravelSelection" DM="No" BY="C" ST="18"/>
    <CH DP="ModemSelection" DM="Internal 56K" BY="D" ST="34"/>
  </CHOICES>
</CONFIGURATION>
```

See Also

[Chapter 12, “Using the COP Java API,” page 161](#)

[Chapter 13, “Understanding the Configurator XML Interface,” Updating a Configuration Interactively, page 186](#)

Domain Member Data

The CONTROL_DATA element, DP element, ATTR element, and attributes of these elements let you retrieve information on domain members.

Every Decision Point

The CONTROL_DATA element lets you retrieve every domain member for every decision point (selection point) in the configuration by decision point.

Including in the REQUEST

The CONTROL_DATA element without an attribute.

Causes the COP to return in the RESPONSE

Every DM element for every DP element in the configuration. Each DP element represents a decision point in the configuration. Each DM element is a child element of a DP element, and represents a domain member of that decision point.

For example:

REQUEST

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA/>
</CONFIGURATION>
```

RESPONSE

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILE_VERSION="20001030-=>
102408-606">
  <CONTROL_DATA>
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower">
      </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower">
      </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower">
      </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC">
      </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC">
      </DM>
    </DP>
    <DP NM="HardDrivesSelection">
      <DM NM="4.3 GB BATRA-33">
      </DM>
      <DM NM="8.4 GB BATA-33">
      </DM>
      <DM NM="13.6 GB BATA-33">
      </DM>
      <DM NM="Quantum 27.3 GB SCSI">
      </DM>
    </DP>
    <DP NM="DVDorCDRomSelection">
      <DM NM="Pioneer DVD-A115">
      </DM>
      <DM NM="Sharp DVD-A100U">
      </DM>
      <DM NM="48X Max Variable CD-ROM">
      </DM>
    </DP>
    <DP NM="...">
      <DM NM="...">
```

```

    </DM>
  </DP>
  . . . .
</CONTROL_DATA>
</CONFIGURATION>

```

Selected Decision Points

The CONTROL_DATA element, its child element DP, and the NM attribute of the DP element let you retrieve every domain member for a selected decision point in the configuration.

A Single Decision Point

Including in the REQUEST the CONTROL_DATA element, and its child element DP with a valid value (decision point name) for the NM attribute of DP causes the COP to return in the RESPONSE every DM element for the named DP element. Each DM is a child element of the named DP, and represents a domain member of that decision point.

For example:

REQUEST

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA>
    <DP NM="HardDrivesSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILE_VERSION="20001030-=>
102408-606">
  <CONTROL_DATA>
    <DP NM="HardDrivesSelection">
      <DM NM="4.3 GB BATRA-33">
        </DM>
      <DM NM="8.4 GB BATA-33">
        </DM>
      <DM NM="13.6 GB BATA-33">
        </DM>
      <DM NM="Quantum 27.3 GB SCSI">
        </DM>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>

```

Multiple Decision Points

The request may include more than one DP element, each with a valid value (decision point name) for its NM attribute:

REQUEST

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA>

```

```

    <DP NM="ScannersSelection"/>
    <DP NM="PrintersSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILE_VERSION="20001030-=>
102408-606">
  <CONTROL_DATA>
    <DP NM="PrintersSelection">
      <DM NM="HP Laserjet">
        </DM>
      <DM NM="Epson 1500">
        </DM>
    </DP>
    <DP NM="ScannersSelection">
      <DM NM="NEC Technologies PediScan">
        </DM>
      <DM NM="Fujitsu ScanPartner 15C">
        </DM>
      <DM NM="Canon DR5080C">
        </DM>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>

```

No Decision Points

Including in the REQUEST the CONTROL_DATA element and its child element DP with an empty string value for the NM attribute of DP, causes the COP to return in the RESPONSE no DM elements and no DP elements.

For example:

REQUEST

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA>
    <DP NM="" />
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILE_VERSION="20001030-=>
102408-606">
  <CONTROL_DATA>
  </CONTROL_DATA>
</CONFIGURATION>

```

Sorting Domain Members

The DMSORT_ST attribute of the CONTROL_DATA element lets you sort domain members returned in the response, by state.

Including in the REQUEST the CONTROL_DATA element with its DMSORT_ST attribute set to “true” causes the COP to return in the RESPONSE domain members (DM’s) sorted by state (ST).

For example:

REQUEST (with a request to sort domain members by state)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="PowerCordSelection" BY="U" QTY="1" DM="Power Brick Cord"/>
  </CHOICES>
  <CONTROL_DATA DMSORT_ST="true">
    <DP NM="BaseSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>
```

RESPONSE (with domain members sorted by state)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215=>
134209-537">
  <CONTROL_DATA>
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="83">
    </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC">
    </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC">
    </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower" ST="17">
    </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower" ST="17">
    </DM>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>
```

Example using the same request and response but without sorting:

REQUEST (with no request to sort domain members by state)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="PowerCordSelection" BY="U" QTY="1" DM="Power Brick Cord"/>
  </CHOICES>
  <CONTROL_DATA>
    <DP NM="BaseSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>
```

RESPONSE 2 (with domain members not sorted by state)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215=>
134209-537">
  <CONTROL_DATA>
```

```

<DP NM="BaseSelection">
  <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="83">
  </DM>
  <DM NM="Suprema Base Pentium III 750 Mhz MidTower" ST="17">
  </DM>
  <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower" ST="17">
  </DM>
  <DM NM="Advanta Base TM3120 400 Mhz Mobile PC">
  </DM>
  <DM NM="Extra Base Tm5400 700 Mhz Mobile PC">
  </DM>
</DP>
</CONTROL_DATA>
</CONFIGURATION>

```

Note. A custom sort may be installed using COP Extensions.

Filtering Domain Members

The `FILTER_EL`, `FILTER_LO`, and `FILTER_HI` attributes of the `CONTROL_DATA` element let you filter domain members returned in the response, by elimination level.

Including in the REQUEST	Causes the COP to return in the RESPONSE
<p>The <code>DECISION_POINTS</code> element, with its <code>ALL</code> attribute set to "false."</p> <ul style="list-style-type: none"> • The <code>CONTROL_DATA</code> element with: • The <code>DP</code> element. <ul style="list-style-type: none"> - Its <code>FILTER_EL</code> attribute set to "true". - Its <code>FILTER_EL_LO</code> and <code>FILTER_EL_HI</code> attributes set to valid elimination levels. 	<p>DM elements whose elimination levels are between <code>FILTER_EL_LO</code> and <code>FILTER_EL_HI</code>, inclusive.</p>

For example:

This REQUEST

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA FILTER_EL="true" FILTER_EL_LO="2" FILTER_EL_HI="5">
    <DP NM="HardDrivesSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

returns a `RESPONSE` that includes only those DM elements for the `HardDrives` DP element that have elimination levels between 2 and 5, inclusive.

Explanations

The `EXPLANATIONS` attribute of the `CONTROL_DATA` element lets you retrieve constraint violation explanations for decisions points in the configuration.

See [Chapter 16, "Retrieving Configuration Information," Global Explanations, page 221](#).

Constraint Violations

Including in the REQUEST the CONTROL_DATA element with its EXPLANATIONS attribute set to “true” causes the COP to return in the RESPONSE all EXPLANATION elements for every decision point (DP element) returned. Each EXPLANATION element is a child element of a DP element, and its content is a textual explanation of the constraint violation.

To return an explanation, the request must include incompatible picks—a constraint violation—for a specified decision point.

For example:

REQUEST (with two incompatible picks)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="PowerCordSelection" BY="U" QTY="1" DM="Power Brick Cord"/>
  </CHOICES>
  <CONTROL_DATA EXPLANATIONS="true">
    <DP NM="BaseSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>
```

RESPONSE (with a constraint violation explanation for the specified decision point)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-⇒
134209-537">
  <CONTROL_DATA EXPLANATIONS="true">
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="83">
        </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower" ST="17">
        </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower" ST="17">
        </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC">
        </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC">
        </DM>
      <EXPLANATION>The C433Mini base requires the 10Foot power cord and Mini chassis.⇒
    </EXPLANATION>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>
```

An example of a constraint violation explanation for a specified decision point:

REQUEST (with two incompatible picks)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="ChassisSelection" BY="U" QTY="1" DM="Full Tower"/>
  </CHOICES>
  <CONTROL_DATA EXPLANATIONS="true">
```

```

    <DP NM="BaseSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE (with a constraint violation explanation for the specified decision point)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-=>
134209-537">
  <CONTROL_DATA EXPLANATIONS="true">
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="83">
        </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower" ST="17">
        </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower">
        </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC" ST="17">
        </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC" ST="17">
        </DM>
      <EXPLANATION>The C433Mini base requires the 10Foot power cord and Mini chassis.</EXPLANATION>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>

```

No Constraint Violation

If the request includes compatible picks, there are no constraint violations, and asking for explanations will not return any explanations.

For example:

REQUEST (with two compatible picks)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="PowerCordSelection" BY="U" QTY="1" DM="10 Foot PC Power Cord"/>
  </CHOICES>
  <CONTROL_DATA EXPLANATIONS="true">
    <DP NM="BaseSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE (without any explanations for the specified decision point)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-=>
134209-537">
  <CONTROL_DATA EXPLANATIONS="true">
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="66">
        </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower" ST="17">
        </DM>

```

```

    <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower" ST="17">
  </DM>
  <DM NM="Advanta Base TM3120 400 Mhz Mobile PC" ST="17">
  </DM>
  <DM NM="Extra Base Tm5400 700 Mhz Mobile PC" ST="17">
  </DM>
</DP>
</CONTROL_DATA>
</CONFIGURATION>

```

An example of a request for explanations failing to return any explanations for the specified decision point, because the picks are compatible:

REQUEST (with two compatible picks)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="ChassisSelection" BY="U" QTY="1" DM="Mini Tower"/>
  </CHOICES>
  <CONTROL_DATA EXPLANATIONS="true">
    <DP NM="BaseSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE (without any explanations for the specified decision point)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215=>
134209-537">
  <CONTROL_DATA EXPLANATIONS="true">
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="66">
      </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower" ST="17">
      </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower" ST="17">
      </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC" ST="17">
      </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC" ST="17">
      </DM>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>

```

Attributes

The ATTR element and its NM attribute let you retrieve the values for a selected attribute for either:

- Every domain member returned in the response.
- Selected domain members returned in the response.

Either way, the response includes the values for the selected attribute for all of the appropriate domain members, by domain member.

Every Domain Member

The ATTR element and its NM attribute let you retrieve the values for a named attribute for every domain member of every decision point returned in the response.

Including in the REQUEST the CONTROL_DATA element and its child element ATTR with a valid value for the NM attribute of the ATTR element causes the COP to return in the RESPONSE:

- The named ATTR element for every DM element of every DP element returned in the response.
- Attribute-value content for each ATTR element of every returned DM element that has the named attribute.

In the following example, all domain members of both decision points—Scanners, and Printers—have the ShortName attribute:

REQUEST (requesting values for the ShortName attribute for every domain member of two decision points)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA>
    <ATTR NM="ShortName"/>
    <DP NM="ScannersSelection"/>
    <DP NM="PrintersSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>
```

RESPONSE (with ShortName attribute-value content for every domain member of both decision points)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILER_VERSION="20001030-=>
102408-606">
  <CONTROL_DATA>
    <DP NM="PrintersSelection">
      <DM NM="HP Laserjet">
        <ATTR NM="SHORTNAME">HPLJ</ATTR>
      </DM>
      <DM NM="Epson 1500">
        <ATTR NM="SHORTNAME">Ep1500</ATTR>
      </DM>
    </DP>
    <DP NM="ScannersSelection">
      <DM NM="NEC Technologies PediScan">
        <ATTR NM="SHORTNAME">NECPeSc</ATTR>
      </DM>
      <DM NM="Fujitsu ScanPartner 15C">
        <ATTR NM="SHORTNAME">FScP15C</ATTR>
      </DM>
      <DM NM="Canon DR5080C">
        <ATTR NM="SHORTNAME">CDR5080C</ATTR>
      </DM>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>
```

In the following example, all domain members of the Base decision point have only the SystemType attribute, whereas all domain members of the Monitors decision point have only the Watts attribute:

REQUEST (requesting values for two attributes for every domain member of two decision points)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
  </CHOICES>
  <CONTROL_DATA>
    <ATTR NM="Watts"/>
    <ATTR NM="SystemType"/>
    <DP NM="BaseSelection"/>
    <DP NM="MonitorsSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE (with SystemType attribute-value content for every domain member of the Base decision point, and Watts attribute-value content for every domain member of the Monitors decision point)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215=>
134209-537">
  <CONTROL_DATA>
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="66">
        <ATTR NM="WATTS"></ATTR>
        <ATTR NM="SYSTEMTYPE">LowEnd</ATTR>
      </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower">
        <ATTR NM="WATTS"></ATTR>
        <ATTR NM="SYSTEMTYPE">MidLevel</ATTR>
      </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower">
        <ATTR NM="WATTS"></ATTR>
        <ATTR NM="SYSTEMTYPE">HighEnd</ATTR>
      </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC">
        <ATTR NM="WATTS"></ATTR>
        <ATTR NM="SYSTEMTYPE">Laptop</ATTR>
      </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC">
        <ATTR NM="WATTS"></ATTR>
        <ATTR NM="SYSTEMTYPE">Laptop</ATTR>
      </DM>
    </DP>
    <DP NM="MonitorsSelection">
      <DM NM="15 Inch Monitor" ST="34">
        <ATTR NM="WATTS">1.0</ATTR>
        <ATTR NM="SYSTEMTYPE"></ATTR>
      </DM>
      <DM NM="17 Inch Monitor">
        <ATTR NM="WATTS">3.0</ATTR>
        <ATTR NM="SYSTEMTYPE"></ATTR>
      </DM>
      <DM NM="21 Inch Monitor">
        <ATTR NM="WATTS">5.0</ATTR>

```

```

    <ATTR NM="SYSTEMTYPE"></ATTR>
  </DM>
</DP>
</CONTROL_DATA>
</CONFIGURATION>

```

Selected Domain Members

The ATTR element and its NM attribute let you retrieve the values for a named attribute for every domain member of selected decision points returned in the response.

Including in the REQUEST

The DECISION_POINTS element, with its ALL attribute set to "false."

- The CONTROL_DATA element.
- The DP element.
- The ATTR element (as a child of the DP element) with a valid value for its NM attribute.

Causes the COP to return in the RESPONSE

- The named ATTR element for every DM element of the selected DP element.
- Attribute-value content for each ATTR element of every returned DM element that has the named attribute.

REQUEST (requesting values for the ShortName attribute for every domain member of the Printers decision point)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA>
    <DP NM="ScannersSelection"/>
    <DP NM="PrintersSelection">
      <ATTR NM="ShortName"/>
    </DP>
    <DP NM="SpeakersSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE (with ShortName attribute-value content for every domain member of the Printers decision point)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILE_VERSION="20001030-⇒
102408-606">
  <CONTROL_DATA>
    <DP NM="SpeakersSelection">
      <DM NM="Altec Lansing 2000">
        </DM>
      <DM NM="Cambridge SoundWorks SP">
        </DM>
    </DP>
    <DP NM="PrintersSelection">
      <DM NM="HP Laserjet">
        <ATTR NM="SHORTNAME">HPLJ</ATTR>
      </DM>
      <DM NM="Epson 1500">
        <ATTR NM="SHORTNAME">Ep1500</ATTR>
      </DM>
    </DP>

```

```

    <DP NM="ScannersSelection">
      <DM NM="NEC Technologies PediScan">
        </DM>
      <DM NM="Fujitsu ScanPartner 15C">
        </DM>
      <DM NM="Canon DR5080C">
        </DM>
      </DP>
    </CONTROL_DATA>
  </CONFIGURATION>

```

Combining Requests

The request may include both types of requests for the values for selected domain member attributes.

The following example requests the values for these domain member attributes:

- Watts—for every domain member of every decision point.
- ShortName—for every domain member of the Printers, and Speakers decision points.
- Description—for every domain member of the Printers decision point.

REQUEST

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA>
    <ATTR NM="Watts"/>
    <DP NM="ScannersSelection"/>
    <DP NM="PrintersSelection">
      <ATTR NM="Description"/>
      <ATTR NM="ShortName"/>
    </DP>
    <DP NM="SpeakersSelection">
      <ATTR NM="ShortName"/>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILE_VERSION="20001030-=>
102408-606">
  <CONTROL_DATA>
    <DP NM="SpeakersSelection">
      <DM NM="Altec Lansing 2000">
        <ATTR NM="SHORTNAME">AL2000</ATTR>
        <ATTR NM="WATTS">1.0</ATTR>
      </DM>
      <DM NM="Cambridge SoundWorks SP">
        <ATTR NM="SHORTNAME">CSWSP</ATTR>
        <ATTR NM="WATTS">1.0</ATTR>
      </DM>
    </DP>
    <DP NM="PrintersSelection">

```

```

<DM NM="HP Laserjet">
  <ATTR NM="DESCRIPTION">HP Laserjet</ATTR>
  <ATTR NM="SHORTNAME">HPLJ</ATTR>
  <ATTR NM="WATTS">3.0</ATTR>
</DM>
<DM NM="Epson 1500">
  <ATTR NM="DESCRIPTION">Epson 1500</ATTR>
  <ATTR NM="SHORTNAME">Ep1500</ATTR>
  <ATTR NM="WATTS">3.0</ATTR>
</DM>
</DP>
<DP NM="ScannersSelection">
  <DM NM="NEC Technologies PediScan">
    <ATTR NM="WATTS">4.0</ATTR>
  </DM>
  <DM NM="Fujitsu ScanPartner 15C">
    <ATTR NM="WATTS">4.0</ATTR>
  </DM>
  <DM NM="Canon DR5080C">
    <ATTR NM="WATTS">4.0</ATTR>
  </DM>
</DP>
</CONTROL_DATA>
</CONFIGURATION>

```

Delta Price

The DPR attribute of the DP element lets you retrieve the delta prices for domain members of a selected decision point.

Including in the REQUEST

The DECISION_POINTS element, with its ALL attribute set to “false.”

- The CONTROL_DATA element.
- The child element DP with:
 - A valid value (decision point name) for its NM attribute.
 - Its DPR attribute set to “true”.

Causes the COP to return in the RESPONSE

The delta price for each DM element of the named DP element, included as the value for the PR attribute of each DM element.

For example:

REQUEST (requesting delta pricing)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA>
    <DP NM="ScannersSelection" DPR="true"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE (with delta pricing as the value for each PR)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215=>
134209-537">
  <CONTROL_DATA>
    <DP NM="ScannersSelection" DPR="true">
      <DM NM="NEC Technologies PediScan" PR="-25.0">
        </DM>
      <DM NM="Fujitsu ScanPartner 15C" PR="0.0">
        </DM>
      <DM NM="Canon DR5080C" PR="35.0">
        </DM>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>

```

Class

The CL attribute of the DP element lets you retrieve the class name for domain members of a selected decision point.

Including in the REQUEST

The DECISION_POINTS element, with its ALL attribute set to "false."

- The CONTROL_DATA element.
- The child element DP with:
 - A valid value (decision point name) for its NM attribute.
 - An empty string value for its CL attribute.

Causes the COP to return in the RESPONSE

The class name for each DM element of the named DP element, included as the value for the CL attribute of each DM element.

For example:

REQUEST

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  , <CONTROL_DATA>
    <DP NM="ScannersSelection" CL=""/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215=>
134209-537">
  <CONTROL_DATA>
    <DP NM="ScannersSelection">
      <DM NM="NEC Technologies PediScan" CL="Scanners">
        </DM>
      <DM NM="Fujitsu ScanPartner 15C" CL="Scanners">
        </DM>
      <DM NM="Canon DR5080C" CL="Scanners">
        </DM>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>

```

```

    </DM>
  </DP>
</CONTROL_DATA>
</CONFIGURATION>

```

State and Quantity

The DP element without any special attributes lets you retrieve the state and quantity of domain members of a selected decision point.

Including in the REQUEST

The CONTROL_DATA element.

- The CONTROL_DATA element.
- The child element DP with a valid value (decision point name) for its NM attribute.

Causes the COP to return in the RESPONSE

Non-null values for the state (ST) and quantity (QTY) attributes of all domain members (DM elements) of the named decision point (DP elements).

Note. The request should not include the ST and QTY attributes for the named DP elements.

For example:

REQUEST (without any special DP attributes)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES RET="false">
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
  </CHOICES>
  <CONTROL_DATA>
    <DP NM="BaseSelection"/>
  </CONTROL_DATA>
</CONFIGURATION>

```

RESPONSE (with a non-null value for ST)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-0" COMPILER_VERSION="20001030=>
102408-606">
  <CONTROL_DATA>
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="66">
        </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower">
        </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower">
        </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC">
        </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC">
        </DM>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>

```

Multi-Select Decision Points

Specifics of the request/response are:

Including in the REQUEST	Causes the COP to return in the RESPONSE
<p>The DECISION_POINTS element, with its ALL attribute set to "false."</p> <ul style="list-style-type: none"> • The CONTROL_DATA element. • The DP element with: <ul style="list-style-type: none"> - A valid value (decision point name) for its NM attribute. - Its MS attribute set to "true". 	<p>Whether or not the named decision point is multi-select. If the decision point (DP element) is multi-select, its MS attribute is set to "true"; if the decision point is single-select, it has no MS attribute.</p>

For example:

REQUEST (asking whether two decision points are multi-select)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CONTROL_DATA>
    <DP NM="ScannersSelection" MS="true"/>
    <DP NM="DeviceControllersSelection" MS="true"/>
  </CONTROL_DATA>
</CONFIGURATION>
```

RESPONSE (answering that only the DeviceControllers decision point is multi-select)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-=>
134209-537">
  <CONTROL_DATA>
    <DP NM="DeviceControllersSelection" MS="true">
      <DM NM="IDE Cable">
        </DM>
      <DM NM="SCSI Cable">
        </DM>
      <DM NM="ATAPI">
        </DM>
    </DP>
    <DP NM="ScannersSelection">
      <DM NM="NEC Technologies PediScan">
        </DM>
      <DM NM="Fujitsu ScanPartner 15C">
        </DM>
      <DM NM="Canon DR5080C">
        </DM>
    </DP>
  </CONTROL_DATA>
</CONFIGURATION>
```

Global Explanations

The EXPLANATIONS attribute of the VIOLATIONS element lets you retrieve constraint violation explanations for the configuration as a whole.

See Also

[Chapter 16, “Retrieving Configuration Information,” Explanations, page 209](#)

Global Only

Including in the REQUEST the VIOLATIONS element with its EXPLANATIONS attribute set to “true” causes the COP to return in the RESPONSE all EXPLANATION elements for the configuration. Each EXPLANATION element is a child element of a VIOLATIONS element, and its content is a textual explanation of the constraint violation.

To return an explanation, the request must include incompatible picks—a constraint violation—for the configuration.

For example:

REQUEST (with two incompatible picks)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="PowerCordSelection" BY="U" QTY="1" DM="Power Brick Cord"/>
  </CHOICES>
  <CONTROL_DATA>
    <DP NM="BaseSelection"/>
  </CONTROL_DATA>
  <VIOLATIONS EXPLANATIONS="true"/>
</CONFIGURATION>
```

RESPONSE (with a global explanation of the constraint violation)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-⇒
134209-537">
  <CONTROL_DATA>
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="83">
        </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower" ST="17">
        </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower" ST="17">
        </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC">
        </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC">
        </DM>
    </DP>
  </CONTROL_DATA>
  <VIOLATIONS>
```

```

    <EXPLANATION>The C433Mini base requires the 10Foot power cord and Mini chassis.=>
  </EXPLANATION>
</VIOLATIONS>
</CONFIGURATION>

```

Global and Decision Point

The request and response can include explanations for both decision points and the configuration.

See [Chapter 16, “Retrieving Configuration Information,” Explanations, page 209](#).

For example: REQUEST (with two incompatible picks)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="PowerCordSelection" BY="U" QTY="1" DM="Power Brick Cord"/>
  </CHOICES>
  <CONTROL_DATA EXPLANATIONS="true">
    <DP NM="BaseSelection"/>
  </CONTROL_DATA>
  <VIOLATIONS EXPLANATIONS="true"/>
</CONFIGURATION>

```

RESPONSE (with an explanation for the specified decision point and a global explanation)

```

<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-=>
134209-537">
  <CONTROL_DATA EXPLANATIONS="true">
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="83">
        </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower" ST="17">
        </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower" ST="17">
        </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC">
        </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC">
        </DM>
      <EXPLANATION>The C433Mini base requires the 10Foot power cord and Mini chassis.=>
    </EXPLANATION>
    </DP>
  </CONTROL_DATA>
  <VIOLATIONS>
    <EXPLANATION>The C433Mini base requires the 10Foot power cord and Mini chassis.=>
  </EXPLANATION>
  </VIOLATIONS>
</CONFIGURATION>

```

Numeric Values

The NUMERIC_VALUES element, the child element NUM, and the NM attribute of the NUM element let you retrieve numeric values for the configuration.

All Values

Including in the REQUEST the NUMERIC_VALUES element causes the COP to return in the RESPONSE all NUM elements for the configuration. Each NUM element is a child element of NUMERIC_VALUES that has attribute-value pairs for its NM and VL attributes.

For example:

REQUEST

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
  </CHOICES>
  <NUMERIC_VALUES/>
</CONFIGURATION>
```

RESPONSE

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-⇒
134209-537">
  <NUMERIC_VALUES>
    <NUM NM="[_Application].WattsSummation" VL="3.0" >
  </NUM>
    <NUM NM="[_Application].StorageCapacitySummation" VL="4300" >
  </NUM>
    <NUM NM="[_Application].PCISlotsResourceBalancing-Provider" VL="3" >
  </NUM>
    <NUM NM="[_Application].PCISlotsResourceBalancing-Consumer" VL="1" >
  </NUM>
    <NUM NM="[_Application].AGPSlotsREsourceBalancing-Provider" VL="0" >
  </NUM>
    <NUM NM="[_Application].AGPSlotsREsourceBalancing-Consumer" VL="0" >
  </NUM>
    <NUM NM="[_Application].ISASlotsResourceBalancing-Provider" VL="4" >
  </NUM>
    <NUM NM="[_Application].ISASlotsResourceBalancing-Consumer" VL="0" >
  </NUM>
  </NUMERIC_VALUES>
</CONFIGURATION>
```

Selected Values

Including in the REQUEST the NUMERIC_VALUES element, and the child element NUM with a valid value for its NM attribute causes the COP to return in the RESPONSE the NUMERIC_VALUES element, and as a child element, the NUM element named in the request with a value for its VL attribute.

For example:

REQUEST

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
  </CHOICES>
  <NUMERIC_VALUES>
    <NUM NM="[_Application].WattsSummation"/>
  </NUMERIC_VALUES>
</CONFIGURATION>
```

RESPONSE

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-=>
134209-537">
  <NUMERIC_VALUES>
    <NUM NM="[_Application].WattsSummation" VL="3.0" >
  </NUM>
</NUMERIC_VALUES>
</CONFIGURATION>
```

Value (VL)

The value for the VL attribute for the named NUM element is the numeric value for the configuration. It is the total value for the named NUM element for all picks in the configuration that have the element as an attribute.

For example:

REQUEST 1 (with one user pick)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
  </CHOICES>
  <CONTROL_DATA>
    <ATTR NM="Watts"/>
    <DP NM="BaseSelection"/>
    <DP NM="MonitorsSelection"/>
    <DP NM="HardDrivesSelection"/>
  </CONTROL_DATA>
  <NUMERIC_VALUES>
    <NUM NM="[_Application].WattsSummation"/>
  </NUMERIC_VALUES>
</CONFIGURATION>
```

RESPONSE 1 (with the one user pick and two computer picks)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-=>
134209-537">
  <CONTROL_DATA>
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="66" >
      <ATTR NM="WATTS"></ATTR>
```

```

</DM>
<DM NM="Suprema Base Pentium III 750 Mhz MidTower" >
  <ATTR NM="WATTS"></ATTR>
</DM>
<DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower" >
  <ATTR NM="WATTS"></ATTR>
</DM>
<DM NM="Advanta Base TM3120 400 Mhz Mobile PC" >
  <ATTR NM="WATTS"></ATTR>
</DM>
<DM NM="Extra Base Tm5400 700 Mhz Mobile PC" >
  <ATTR NM="WATTS"></ATTR>
</DM>
</DP>
<DP NM="HardDrivesSelection">
  <DM NM="4.3 GB BATRA-33" ST="34" >
    <ATTR NM="WATTS">2.0</ATTR>
  </DM>
  <DM NM="8.4 GB BATA-33" >
    <ATTR NM="WATTS">2.0</ATTR>
  </DM>
  <DM NM="13.6 GB BATA-33" >
    <ATTR NM="WATTS">2.0</ATTR>
  </DM>
  <DM NM="Quantum 27.3 GB SCSI" >
    <ATTR NM="WATTS">2.0</ATTR>
  </DM>
</DP>
<DP NM="MonitorsSelection">
  <DM NM="15 Inch Monitor" ST="34" >
    <ATTR NM="WATTS">1.0</ATTR>
  </DM>
  <DM NM="17 Inch Monitor" >
    <ATTR NM="WATTS">3.0</ATTR>
  </DM>
  <DM NM="21 Inch Monitor" >
    <ATTR NM="WATTS">5.0</ATTR>
  </DM>
</DP>
</CONTROL_DATA>
<NUMERIC_VALUES>
  <NUM NM="[_Application].WattsSummation" VL="3.0" >
  </NUM>
</NUMERIC_VALUES>
</CONFIGURATION>

```

The value for `[_Application].WattsSummation` is 3.0—the total watts for all picks in the configuration that have WATTS as an attribute. The user pick Prima Base Celeron 433Mhz MiniTower does not have WATTS as an attribute. Two computer picks required by the user pick have WATTS as an attribute. The total value for the WATTS attribute for those picks is 3.0:

- 2.0 watts for the 4.3 GB BATRA-33 hard drive pick, plus
- 1.0 watts for the 15 Inch Monitor pick .

In the following example, there are two user picks and one computer pick. The total value (VL) for the WATTS attribute for all picks in the configuration having the WATTS attribute is 7.0:

REQUEST 2 (with two user picks)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="">
  <CHOICES>
    <CH DP="BaseSelection" BY="U" QTY="1" DM="Prima Base Celeron 433Mhz MiniTower"/>
    <CH DP="MonitorsSelection" BY="U" QTY="1" DM="21 Inch Monitor"/>
  </CHOICES>
  <CONTROL_DATA>
    <ATTR NM="Watts"/>
    <DP NM="BaseSelection"/>
    <DP NM="MonitorsSelection"/>
    <DP NM="HardDrivesSelection"/>
  </CONTROL_DATA>
  <NUMERIC_VALUES>
    <NUM NM="[ _Application ].WattsSummation"/>
  </NUMERIC_VALUES>
</CONFIGURATION>
```

RESPONSE 2 (with two user picks and one computer pick)

```
<CONFIGURATION MODEL_ID="Sample" MODEL_VERSION="1-0-1" COMPILE_VERSION="20001215-=>
134209-537">
  <CONTROL_DATA>
    <DP NM="BaseSelection">
      <DM NM="Prima Base Celeron 433Mhz MiniTower" ST="66" >
        <ATTR NM="WATTS"></ATTR>
      </DM>
      <DM NM="Suprema Base Pentium III 750 Mhz MidTower" >
        <ATTR NM="WATTS"></ATTR>
      </DM>
      <DM NM="Magna Base Pentium III Xeon 100 Mhz FullTower" >
        <ATTR NM="WATTS"></ATTR>
      </DM>
      <DM NM="Advanta Base TM3120 400 Mhz Mobile PC" >
        <ATTR NM="WATTS"></ATTR>
      </DM>
      <DM NM="Extra Base Tm5400 700 Mhz Mobile PC" >
        <ATTR NM="WATTS"></ATTR>
      </DM>
    </DP>
    <DP NM="HardDrivesSelection">
      <DM NM="4.3 GB BATRA-33" ST="34" >
        <ATTR NM="WATTS">2.0</ATTR>
      </DM>
      <DM NM="8.4 GB BATA-33" >
        <ATTR NM="WATTS">2.0</ATTR>
      </DM>
    </DP>
  </CONTROL_DATA>
```

```
<DM NM="13.6 GB BATA-33" >
  <ATTR NM="WATTS">2.0</ATTR>
</DM>
<DM NM="Quantum 27.3 GB SCSI" >
  <ATTR NM="WATTS">2.0</ATTR>
</DM>
</DP>
<DP NM="MonitorsSelection">
  <DM NM="15 Inch Monitor" >
    <ATTR NM="WATTS">1.0</ATTR>
  </DM>
  <DM NM="17 Inch Monitor" >
    <ATTR NM="WATTS">3.0</ATTR>
  </DM>
  <DM NM="21 Inch Monitor" ST="66" >
    <ATTR NM="WATTS">5.0</ATTR>
  </DM>
</DP>
</CONTROL_DATA>
<NUMERIC_VALUES>
  <NUM NM="[_Application].WattsSummation" VL="7.0" >
  </NUM>
</NUMERIC_VALUES>
</CONFIGURATION>
```


CHAPTER 17

Retrieving Saved Configuration Information

This chapter describes how to retrieve configuration details using the XML request and provides sample snippets of both the request and response for the different details that can be retrieved. This chapter has these sections:

- Elements and Attributes
- The CONFIGURATION element
- The CONFIG_DETAILS element
- The SECTION element
- Total Price
- Compound Violations
- Components
- Choices
- Choice Violations
- Component Violations
- Externs
- Numeric Values
- External Variables
- Configuration Attributes
- Hierarchical Component Structure
- Connections
- Completeness Information
- Summary, Configuration Information Elements and Attributes

Understanding Saved Configuration Information

The COPXML ConfigDetails feature lets you retrieve details about a saved configuration, including:

- Total price for the configuration.
- Delta information for changes between two versions of a configuration.
- Information regarding configuration validity and violations.
- Any existing conflicts in the configuration whether at the compound, component, or choice level.
- Completeness information.

- Component data, when returned, can include:
 - Name, type, ID, Total Price, SolveDate.
 - Choices—All choices, those specified by name, or those filtered by an attribute value. Selection point name, domain member name, state, and quantity (returned by default). Specified attribute values for each selection, optionally mapped to a different attribute name.
 - Externs—All or only those specified by name. Name and value(s). Values can be a collection of one or more.
 - Numeric Values—All or only those specified by name. Name, value, and type.
 - Config attributes—All or only those specified by name. Name and value.
- Hierarchical component structure for compounds.
 - Name, type, ID, Total Price, SolveDate.
- Connection data for compounds, when returned, includes:
 - Name, type, and ID.
 - Name, type, and ID of the connecting (from) component.
 - Name, type, and ID of the connected (to) component.

Note. The ConfigDetails request is processed separately from and supersedes any other elements included in the same request. Thus, if the CONFIG_DETAILS element is included in the request, the only operation performed during that post to the servlet is the configuration details request. Any other elements/attributes in the request will be ignored.

Elements and Attributes

The COPXML ConfigDetails request may include these elements and attributes to retrieve details about a configuration.

```

CONFIGURATION configId solutionID validate
CONFIG_DETAILS
  FLAG name value
  SECTION nm
  COMPONENTS
    FLAG name value
    COMPONENT_DEFINITION component
    FLAG name value
    VALUE
    FLAG_SET name
    VALUE
  SELECTION_ATTRIBUTES
    SELECTION_POINT name
    ATTRIBUTE name mapto
  STRUCTURE value
  SUBSTRUCTURE type value
  CONNECTIONS
    FLAG name value

```

The COPXML ConfigDetails response may include these elements and attributes to return details about a configuration.

```

CONFIG_DETAILS configId solutionId hasViolations isValid TOTAL_PRICE
VIOLATIONS
  EXPLANATION
COMPOUND_CONFIGURATION name type
SECTION nm
COMPONENTS
  CONFIGURABLE_COMPONENT name component id TOTAL_PRICE
  CONFIGURATION
  VIOLATIONS
    EXPLANATION
  COMPLETE STATUS
  DP NM
  CHOICES
    CH DP DM ST QTY
  ATTR NM
  NUMERIC_VALUES
    NUM NM VL TY
  EXTERN_VARS
    EV NM
    VAL
  CONFIG_ATTRIBUTES
    ATTR NM TY
STRUCTURE
  CONFIGURABLE_COMPONENT name component id solve TOTAL_PRICE
  CONNECTED_COMPONENT
    CONNECTION id name ref fromCompId fromCompName fromCompType
      to⇒
CompId toCompName toCompType
  CONFIGURABLE_COMPONENT name id component
  CONNECTIONS
    CONNECTION id name ref fromCompId fromCompName fromCompType
      toCompId⇒
toCompName toCompType

```

The CONFIGURATION Element

The CONFIGURATION element must be included in the request. There can be only one CONFIGURATION element per request. It has a child element CONFIG_DETAILS and the attributes *configId*, *solutionId* and *validate*.

The CONFIG_DETAILS element is described in the next subsection. The attribute *configId* lets you specify the id of the configuration that you want to retrieve details for.

The attribute *solutionId* defines the solutionId for the configuration.

The attribute *validate* lets you retrieve information about whether the configuration is valid and if it has any violations.

Including in the REQUEST

`<CONFIGURATION configId="1676995129" solutionId="Construction" validate="true">` causes the ConfigDetails to return in the RESPONSE the configID, solutionId, and the boolean values for whether the configuration is valid and whether it has violations.

For example:

```
<CONFIGURATION configId="1676995129" solutionId="Construction" validate="true"> RESPONSE
<CONFIG_DETAILS configId="1676995129" solutionId="Construction" hasViolations="true"
isValid="false">
```

REQUEST

```
<CONFIGURATION configId="1676995129" solutionId="Construction"
validate="true">
```

RESPONSE

```
<CONFIG_DETAILS configId="1676995129" solutionId="Construction"
hasViolations="true" isValid="false">
```

Information about the validity of the configuration and whether it has violations is not returned in the response if the attribute `validate` is not included or if it is set to “false” in the request.

For example:

REQUEST

```
<CONFIGURATION configId="1676995129" solutionId="Construction"
validate="false">
```

OR

```
<CONFIGURATION configId="1676995129" solutionId="Construction">
```

RESPONSE

```
<CONFIG_DETAILS configId="1676995129" solutionId="Construction">
```

The CONFIG_DETAILS Element

The CONFIG_DETAILS element is a child element of the CONFIGURATION element and must be included in the request to return details of a saved configuration. There can be only one CONFIG_DETAILS element for a request.

The CONFIG_DETAILS element lets you define one or more sections for the information to be returned in the response using the SECTION child element. It also lets you define whether you want total price (for component configurations and compounds) and compound conflicts (for compounds) returned in the response using the FLAG child element.

Differences between the currently displayed configuration and another version include:

- Added, changed, or deleted choices.
- Added, changed, or deleted configuration attributes.
- Changed expression values.
- Additions, deletions, and relocation of components and connections in compound models.

The DELTA_INFO Element

The DELTA_INFO element is a child element of the CONFIG_DETAILS element and must be included in the request to return delta details of a saved configuration.

Differences between the currently displayed configuration and another version include:

- Added, changed, or deleted choices.
- Added, changed, or deleted configuration attributes.
- Changed expression values.
- Additions, deletions, and relocation of components and connections in compound models.

By default, the delta information returned is between the most recently saved version of the configuration and the last version of the configuration whose order status was set to submitted. Optionally, the DELTA_INFO element lets you specify a range of dates to retrieve deltas for.

Components

Including the DELTA_INFO element in a request for CONFIG_DETAILS of a component configuration will result in a response with the following structure:

```

CONFIG_DETAILS configId solutionId hasViolations isValid TOTAL_PRICE
SECTION nm ".
DELTA_INFO startDate endDate
COMPONENT id name type modDate
CONFIG_DELTA
MODEL_DELTA
  PREVIOUS modelName modelVersion compileVersion
  CURRENT modelName modelVersion compileVersion
CHOICE_ADDS
  CH DP DM ST QTY
  EVCH DP
  VAL
CHOICE_DELETES
  CH DP DM ST QTY
  EVCH DP
  VAL
CHOICE_CHANGES
  DELTA_CHOICE DP
  PREVIOUS
    CH DP DM ST QTY
    OR
    EVCH DP
    VAL
  CURRENT
    CH DP DM ST QTY
    OR
    EVCH DP
    VAL
CFG_ATTR_ADDS

```

```

ATTR nm val
CFG_ATTR_DELETES
ATTR nm val
CFG_ATTR_CHANGES
DELTA_ATTR nm PREVIOUS CURRENT
EXPR_ADDS
EXPR nm type val
EXPR_DELETES
EXPR nm type val
EXPR_CHANGES
DELTA_EXPR nm PREVIOUS CURRENT

```

Compounds

Including the DELTA_INFO element in a request for CONFIG_DETAILS of a compound configuration will result in a response with the following structure:

```

CONFIG_DETAILS configId solutionId hasViolations isValid TOTAL_PRICE
VIOLATIONS
EXPLANATION
COMPOUND_CONFIGURATION name type
SECTION nm
DELTA_INFO startDate endDate
COMPONENT_ADDS
COMPONENT id name type modDate
COMPONENT_CHANGES
COMPONENT id name type modDate
CONFIG_DELTA
MODEL_DELTA
PREVIOUS modelName modelVersion compileVersion
CURRENT modelName modelVersion compileVersion
CHOICE_ADDS
CH DP DM ST QTY
EVCH DP
VAL
CHOICE_DELETES
CH DP DM ST QTY
EVCH DP
VAL
CHOICE_CHANGES
DELTA_CHOICE DP
PREVIOUS
CH DP DM ST QTY
OR
EVCH DP
VAL
CURRENT
CH DP DM ST QTY
OR
EVCH DP

```

```

        VAL
    CFG_ATTR_ADDS
        ATTR nm val
    CFG_ATTR_DELETES
        ATTR nm val
    CFG_ATTR_CHANGES
        DELTA_ATTR nm PREVIOUS CURRENT
    EXPR_ADDS
        EXPR nm type val
    EXPR_DELETES
        EXPR nm type val
    EXPR_CHANGES
        DELTA_EXPR nm PREVIOUS CURRENT
    COMPONENT_DELETES
        COMPONENT id name type modDate
    CONNECTION_ADDS
        CONN id name type modDate
    CONNECTION_MOVES
        CONN id name type modDate
        CONN_DELTA
            PREVIOUS fromCompId toCompId
            CURRENT fromCompId toCompId
    CONNECTION_DELETES
        CONN id name type modDate

```

The SECTION Element

The SECTION element is a child element of the CONFIG_DETAILS element. There may be one or more SECTION elements for a given request.

Each SECTION element can be used to define the different components and their details, connections and their details (for compounds) and the hierarchical component structure (for compounds) to be returned in the response by using the COMPONENTS, CONNECTIONS and STRUCTURE child elements respectively. There can be only one each of the COMPONENTS, CONNECTIONS and STRUCTURE child elements in each SECTION element and they are optional. The detailed descriptions of the child elements are discussed in further subsections of this section.

The attribute *nm* of the SECTION element lets you define a title for the section that can be used by the application receiving the response in any manner appropriate for the application. The *nm* attribute is optional and the value for the *nm* attribute does not have to be unique across the request, that is, one or more sections may define the same string for *nm*.

For example:

REQUEST

```
<SECTION nm="PackageDetails">
```

RESPONSE

```
<SECTION nm="PackageDetails">
```

Total Price

The FLAG child element of the CONFIG_DETAILS element lets you retrieve the configuration's total price.

Including in the REQUEST FLAG type= "detailsReturned" value="totalPrice" causes the ConfigDetails to return in the RESPONSE the value for the TOTAL_PRICE attribute for the whole configuration as well as the total price for each component in the respective components's section.

For example:

REQUEST

```
<FLAG type="detailsReturned" value="totalPrice"/>
```

RESPONSE

```
<CONFIG_DETAILS configId="1676995129" solutionId="OutputUtil"
TOTAL_PRICE="211200.00">
<COMPONENTS>
<CONFIGURABLE_COMPONENT name="Building-1" component=
  "Building" id="1676995126" =>
TOTAL_PRICE="2120.00"
  solveDate="20010625">
```

Compound Violations

The FLAG child element of the CONFIG_DETAILS element lets you retrieve a compound configuration's violations.

Including in the REQUEST

FLAG type="detailsReturned" value="compoundConflicts" causes the ConfigDetails to return in the RESPONSE the compound violation explanations.

For example:

REQUEST

```
<FLAG type="detailsReturned" value="compoundConflicts"/>
```

RESPONSE

```
<VIOLATIONS>
<EXPLANATION>Component Building has an invalid configuration.</
EXPLANATION>
</VIOLATIONS>
```

Components

The COMPONENTS child element of the SECTION element and its FLAG and COMPONENT_DEFINITION child elements let you retrieve components and their details. There can be only one COMPONENTS element per SECTION element. The FLAG element under COMPONENTS lets you define the component type filter and the COMPONENT_DEFINITION element lets you define what details to return for a component type. There may be zero or more FLAG elements, each defining a different component type filter.

Case 1

Including in the REQUEST both the FLAG and COMPONENT_DEFINITION elements for one or more component types and every type defined in FLAG that has a matching type defined in COMPONENT_DEFINITION causes ConfigDetails to return in the RESPONSE all components of the requested type(s) and the requested details for those components.

For example:

REQUEST

```
<SECTION nm="PackageDetails">
<COMPONENTS>
<FLAG type="componentFilterType" value="Building"/>
<FLAG type="componentFilterType" value="Floor"/>
  <COMPONENT_DEFINITION component="Building">
    <FLAG type="detailsReturned" value="selections"/>
    <FLAG type="detailsReturned" value="externs"/>
    <FLAG type="detailsReturned" value="expressions"/>
    <FLAG type="detailsReturned" value="configAttributes"/>
    <FLAG type="detailsReturned" value="componentConflicts"/>
    <FLAG type="detailsReturned" value="selectionConflicts"/>
  </COMPONENT_DEFINITION>
  <COMPONENT_DEFINITION component="Floor">
    <FLAG type="detailsReturned" value="selections"/>
    <FLAG type="detailsReturned" value="externs"/>
  </COMPONENT_DEFINITION>
```

RESPONSE

```
<SECTION nm="PackageDetails">
<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
  id=>
"1676995126" TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    <CHOICES>
<<---Details--->>
    <VIOLATIONS>
<<---Details--->>
    </VIOLATIONS>
  </CHOICES>
  <NUMERIC_VALUES>
    <<---Details--->>
  </NUMERIC_VALUES>
```

```

<EXTERN_VARS>
  <<---Details--->>
</EXTERN_VARS>
<VIOLATIONS>
  <<---Details--->>
</VIOLATIONS>
  <CONFIG_ATTRIBUTES>
    <<---Details--->>
  </CONFIG_ATTRIBUTES>
</CONFIGURATION>
</CONFIGURABLE_COMPONENT>
<CONFIGURABLE_COMPONENT name="Floor-1" component="Floor"
  id="1676995128"⇒
TOTAL_PRICE="1900.00" solveDate="20010625">
  <CONFIGURATION>
    <CHOICES>
      <<---Details--->>
    </CHOICES>
  <EXTERN_VARS>
    <<---Details--->>
  </EXTERN_VARS>
</CONFIGURATION>
</CONFIGURABLE_COMPONENT>
</COMPONENTS>

```

Case 2

Including in the REQUEST the FLAG elements for one or more component types and COMPONENT_DEFINITION elements for some, but not all, component types defined in the FLAG element causes ConfigDetails to return in the RESPONSE all components of the requested type(s) that have a matching type defined in a COMPONENT_DEFINITION element and the requested details for those components.

For example:

REQUEST

```

<SECTION nm="PackageDetails">
<COMPONENTS>
<FLAG type="componentFilterType" value="Building"/>
<FLAG type="componentFilterType" value="Floor"/>
  <COMPONENT_DEFINITION component="Floor">
    <FLAG type="detailsReturned" value="selections"/>
    <FLAG type="detailsReturned" value="externs"/>
  </COMPONENT_DEFINITION>

```

RESPONSE

```

<SECTION nm="PackageDetails">
<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Floor-1" component="Floor"
    id="1676995128"⇒
TOTAL_PRICE="1900.00" solveDate="20010625">
  <CONFIGURATION>

```

```

    <CHOICES>
      <<---Details--->>
    </CHOICES>
    <EXTERN_VARS>
      <<---Details--->>
    </EXTERN_VARS>
  </CONFIGURATION>
</CONFIGURABLE_COMPONENT>
</COMPONENTS>

```

Note that components for type Building were not returned as there was no corresponding COMPONENT_DEFINITION element supplied in the request.

Case 3

Including in the REQUEST both the FLAG and COMPONENT_DEFINITION elements for one or more component types causes ConfigDetails to return in the RESPONSE all components of the requested type(s) that have a matching type defined in the COMPONENT_DEFINITION element and the requested details for those components. For those components with no matching type in COMPONENT_DEFINITION, the default definition is used to determine what details to return in the response. Every type defined in FLAG does not have a matching type defined in COMPONENT_DEFINITION). However, it defines a COMPONENT_DEFINITION element without specifying a component type (the default definition).

For example:

REQUEST

```

<SECTION nm="PackageDetails">
<COMPONENTS>
<FLAG type="componentFilterType" value="Building"/>
<FLAG type="componentFilterType" value="Floor"/>
  <COMPONENT_DEFINITION component="Building">
    <FLAG type="detailsReturned" value="expressions"/>
  </COMPONENT_DEFINITION>
  <COMPONENT_DEFINITION> (or <COMPONENT_DEFINITION component="">
    <FLAG type="detailsReturned" value="selections"/>
    <FLAG type="detailsReturned" value="externs"/>
    <FLAG type="detailsReturned" value="configAttributes"/>
  </COMPONENT_DEFINITION>

```

RESPONSE

```

<SECTION nm="PackageDetails">
<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id=>
"1676995126" TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    <NUMERIC_VALUES>
      <<---Details--->>
    </NUMERIC_VALUES>
  </CONFIGURATION>
</CONFIGURABLE_COMPONENT>
  <CONFIGURABLE_COMPONENT name="Floor-1" component="Floor"

```

```

    id="1676995128"⇒
TOTAL_PRICE="1900.00" solveDate="20010625">
  <CONFIGURATION>
    <CHOICES>
      <<---Details--->>
    </CHOICES>
    <EXTERN_VARS>
      <<---Details--->>
    </EXTERN_VARS>
    <CONFIG_ATTRIBUTES>
      <<---Details--->>
    </CONFIG_ATTRIBUTES>
  </CONFIGURATION>
</CONFIGURABLE_COMPONENT>
</COMPONENTS>

```

Case 4

Including in the REQUEST only the COMPONENT_DEFINITION elements for one or more component types, but no FLAG elements, causes ConfigDetails to return in the RESPONSE all components of the compound, or the single component in the case of a component configuration. The corresponding COMPONENT_DEFINITION element is used to return the details. If a COMPONENT_DEFINITION does not exist for a component type, the default definition is used to return details. If a default definition does not exist, only high-level details are returned for components of that type.

For example:

REQUEST

```

<SECTION nm="PackageDetails">
<COMPONENTS>
  <COMPONENT_DEFINITION component="Building">
    <FLAG type="detailsReturned" value="expressions"/>
  </COMPONENT_DEFINITION>
  <COMPONENT_DEFINITION> (or <COMPONENT_DEFINITION component="">
    <FLAG type="detailsReturned" value="externs"/>
    <FLAG type="detailsReturned" value="configAttributes"/>
  </COMPONENT_DEFINITION>
  <COMPONENT_DEFINITION component="Room">
    <FLAG type="detailsReturned" value="externs"/>
  </COMPONENT_DEFINITION>

```

RESPONSE

```

<SECTION nm="PackageDetails">
<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id=⇒
"1676995126" TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    <NUMERIC_VALUES>
      <<---Details--->>
    </NUMERIC_VALUES>

```

```

    </CONFIGURATION>
  </CONFIGURABLE_COMPONENT>
  <CONFIGURABLE_COMPONENT name="Floor-1" component="Floor"
    id="1676995128"=>
    TOTAL_PRICE="1900.00" solveDate="20010625">
    <CONFIGURATION>
    <EXTERN_VARS>
    <<---Details--->>
    </EXTERN_VARS>
    <CONFIG_ATTRIBUTES>
    <<---Details--->>
    </CONFIG_ATTRIBUTES>
    </CONFIGURATION>
  </CONFIGURABLE_COMPONENT>
  <CONFIGURABLE_COMPONENT name="Room-1" component="Room"
    id="1676995129"=>
    TOTAL_PRICE="130.00" solveDate="20010625">
    <CONFIGURATION>
    <EXTERN_VARS>
    <<---Details--->>
    </EXTERN_VARS>
    </CONFIGURATION>
  </CONFIGURABLE_COMPONENT>
</COMPONENTS>

```

Note that details for components of type Building and Room were returned based on the corresponding COMPONENT_DEFINITION elements and that details for components of type Floor were based on the default COMPONENT_DEFINITION element.

Case 5

Including in the REQUEST only the FLAG elements for one or more component types, but no COMPONENT_DEFINITION elements, causes ConfigDetails to return in the RESPONSE only high-level details for all components of the requested type(s) defined in the FLAG elements.

For example:

REQUEST

```

<SECTION nm="PackageDetails">
<COMPONENTS>
<FLAG type="componentFilterType" value="Building"/>
< FLAG type="componentFilterType" value="Floor"/>

```

RESPONSE

```

<SECTION nm="PackageDetails">
<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id=>
    "1676995126" TOTAL_PRICE="2120.00" solveDate="20010625">
  </CONFIGURABLE_COMPONENT>
  <CONFIGURABLE_COMPONENT name="Floor-1" component="Floor"
    id="1676995128"=>

```

```
TOTAL_PRICE="1900.00" solveDate="20010625">
</CONFIGURABLE_COMPONENT>
</COMPONENTS>
```

Case 6

Including in the REQUEST neither the FLAG element nor the COMPONENT_DEFINITION element causes ConfigDetails to return in the RESPONSE only high-level details for all components of the compound or the single component in case of a component configuration.

For example:

REQUEST

```
<SECTION nm="PackageDetails">
<COMPONENTS>
```

RESPONSE

```
<SECTION nm="PackageDetails">
<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id=>
"1676995126" TOTAL_PRICE="2120.00" solveDate="20010625">
  </CONFIGURABLE_COMPONENT>
  <CONFIGURABLE_COMPONENT name="Floor-1" component="Floor"
    id="1676995128">
TOTAL_PRICE="1900.00" solveDate="20010625">
  </CONFIGURABLE_COMPONENT>
  <CONFIGURABLE_COMPONENT name="Room-1" component="Room"
    id="1676995129" TOTAL_>
PRICE="1300.00" solveDate="20010625">
  </CONFIGURABLE_COMPONENT>
</COMPONENTS>
```

Note. In the case of a single component configuration, the element FLAG is ignored. The details for the single component are returned in the response as defined in the COMPONENT_DEFINITION.

In the case of a compound configuration, an extra child element of the CONFIG_DETAILS element is included in the response indicating the name and type of the compound.

Choices

The FLAG and FLAG_SET child elements of the COMPONENT_DEFINITION element let you retrieve choices and their details for a component. The SELECTION_ATTRIBUTES child element of COMPONENT_DEFINITION lets you retrieve attributes for choices and also lets you map the attribute name to a different name to be returned in the response.

Case 1

Including in the REQUEST only the FLAG element specifying choices to be returned, causes ConfigDetails to return in the RESPONSE all choices for a component. Details returned are decision point name, domain member name, state, and quantity.

For example:

REQUEST

```
<COMPONENT_DEFINITION component="Building">
  <FLAG type="detailsReturned" value="selections"/>
  - - -
</COMPONENT_DEFINITION
```

RESPONSE

```
<COMPONENTS>
<CONFIGURABLE_COMPONENT name="Building-1" component="Building"
  id="1676995126"⇒
TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    <CHOICES>
      <CH DP="BuildingTypeSelection" DM="Apartment" ST="66" QTY="1.0">
      </CH>
      <CH DP="BuildingColorSelection" DM="Blue" ST="66" QTY="1.0">
      </CH>
      <CH DP="BuildingHeightSelection" DM="300" ST="66" QTY="1.0">
      </CH>
    </CHOICES>
    - - -
  </CONFIGURATION>
</CONFIGURABLE_COMPONENT>
</COMPONENTS>
```

Case 2

Including in the REQUEST the FLAG element specifying choices to be returned and the SELECTION_ATTRIBUTES element specifying the attributes to be returned, causes ConfigDetails to return in the RESPONSE all choices for a component. Details returned are decision point name, domain member name, state, quantity, and the requested attribute names. Please note the use of the mapTo attribute in the example below.

For example:

REQUEST

```
<COMPONENT_DEFINITION component="Building">
  <FLAG type="detailsReturned" value="selections"/>
  - - -
  <SELECTION_ATTRIBUTES>
    <SELECTION_POINT type="BuildingTypeSelection">
      <ATTRIBUTE nm="listPrice" mapto=""/>
      <ATTRIBUTE nm="description" mapto="desc"/>
    </SELECTION_POINT>
    <SELECTION_POINT type="BuildingColorSelection">
      <ATTRIBUTE nm="sku"/>
    </SELECTION_POINT>
  </SELECTION_ATTRIBUTES>
</COMPONENT_DEFINITION
```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126"⇒
    TOTAL_PRICE="2120.00" solveDate="20010625">
    <CONFIGURATION>
      <CHOICES>
        <CH DP="BuildingTypeSelection" DM="Apartment" ST="66" QTY="1.0">
          <ATTR NM="listPrice">2000.0</ATTR>
          <ATTR NM="desc">Apartment</ATTR>
        </CH>
        <CH DP="BuildingColorSelection" DM="Blue" ST="66" QTY="1.0">
          <ATTR NM="sku">CB00255</ATTR>
        </CH>
        <CH DP="BuildingHeightSelection" DM="300" ST="66" QTY="1.0">
        </CH>
      </CHOICES>
    </CONFIGURATION>
  </CONFIGURABLE_COMPONENT>
</COMPONENTS>

```

Note. To return the same attributes for all the choices, specify the SELECTION_POINT element only once, specifying the attributes you want returned and without specifying the type attribute. Specifying mapTo="" is equivalent to not specifying the mapTo attribute.

Case 3

Including in the REQUEST the FLAG element specifying choices to be returned and the FLAG element specifying the type selectionFilterAttribute, causes ConfigDetails to return in the RESPONSE all choices for a component that have the attribute(s) specified in the selectionFilterAttribute filter (the attribute(s) is not returned in the response in this case). Details returned are decision point name, domain member name, state, and quantity.

The selectionFilterAttribute defines an attribute that must exist on the choice for the choice to be returned in the response. Further, if it has a child element VALUE (this is optional), the value specified must match the value of the attribute on that choice for that choice to be returned. If more than one selectionFilterAttribute is specified, all of them must exist and satisfy the equality condition specified by VALUE for that choice to be returned.

For example:

REQUEST

```

<COMPONENT_DEFINITION component="Building">
  <FLAG type="detailsReturned" value="selections"/>
  <FLAG type="selectionFilterAttribute" value="listPrice">
    <VALUE>2000.0</VALUE>
  </FLAG>
</COMPONENT_DEFINITION>

```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126"⇒

```

```
TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    <CHOICES>
      <CH DP="BuildingTypeSelection" DM="Apartment" ST="66" QTY="1.0">
        </CH>
      </CHOICES>
    </CONFIGURATION>
  <CONFIGURABLE_COMPONENT>
<COMPONENTS>
```

Case 4

Including in the REQUEST the FLAG element specifying choices to be returned, the FLAG element specifying the type selectionFilterAttribute, and the SELECTION_ATTRIBUTES element specifying the attributes to be returned causes ConfigDetails to return in the RESPONSE all choices for a component that have the attribute(s) specified in the selectionFilterAttribute filter (the attribute(s) is returned in the response in this case).

Details returned are decision point name, domain member name, state, quantity and the requested attributes.

For example:

REQUEST

```
<FLAG type="detailsReturned" value="selections"/>
<FLAG type="selectionFilterAttribute" value="listPrice">
  <VALUE>2000.0</VALUE>
</FLAG>
<SELECTION_ATTRIBUTES>
  <SELECTION_POINT type="BuildingTypeSelection">
    <ATTRIBUTE nm="listPrice" mapto=""/>
    <ATTRIBUTE nm="description" mapto="desc"/>
  </SELECTION_POINT>
  <SELECTION_POINT type="BuildingColorSelection">
    <ATTRIBUTE nm="sku"/>
  </SELECTION_POINT>
</SELECTION_ATTRIBUTES>
</COMPONENT_DEFINITION
```

RESPONSE

```
<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126">
    TOTAL_PRICE="2120.00" solveDate="20010625">
      <CONFIGURATION>
        <CHOICES>
          <CH DP="BuildingTypeSelection" DM="Apartment" ST="66" QTY="1.0">
            <ATTR NM="listPrice">2000.0</ATTR>

            <ATTR NM="desc">Apartment</ATTR>
          </CH>
        </CHOICES>
      </CONFIGURATION>
    </CONFIGURABLE_COMPONENT>
```

```
<COMPONENTS>
```

Note. To return the same attributes for all the choices, specify the SELECTION_POINT element only once, specifying the attributes you want returned but not the type attribute.

Case 5

Including in the REQUEST the FLAG element specifying choices to be returned and the FLAG_SET element specifying the specific choice(s) to be returned causes ConfigDetails to return in the RESPONSE the requested choices for a component. Details returned are decision point name, domain member name, state, and quantity.

For example:

REQUEST

```
<COMPONENT_DEFINITION component="Building">
  <FLAG type="detailsReturned" value="selections"/>
  <FLAG_SET type="selectionFilterNames">
    <VALUE>BuildingTypeSelection</VALUE>
    <VALUE>BuildingColorSelection</VALUE>
  </FLAG_SET>
</COMPONENT_DEFINITION>
```

RESPONSE

```
V<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126">
    TOTAL_PRICE="2120.00" solveDate="20010625">
    <CONFIGURATION>
      <CHOICES>
        <CH DP="BuildingTypeSelection" DM="Apartment" ST="66" QTY="1.0">
        </CH>
        <CH DP="BuildingColorSelection" DM="Blue" ST="66" QTY="1.0">
        </CH>
      </CHOICES>
    </CONFIGURATION>
  </CONFIGURABLE_COMPONENT>
</COMPONENTS>
```

Case 6

Including in the REQUEST the FLAG element specifying choices to be returned, the FLAG_SET element specifying the specific choice(s) to be returned, and the SELECTION_ATTRIBUTES element specifying the attributes to be returned. causes ConfigDetails to return in the RESPONSE the requested choices for a component. See Cases 2 and 5 above.

Details returned are decision point name, domain member name, state, quantity, and requested attributes.

For example:

REQUEST

```
<COMPONENT_DEFINITION component="Building">
  <FLAG type="detailsReturned" value="selections"/>
  <FLAG_SET type="selectionFilterNames">
```

```

    <VALUE>BuildingTypeSelection</VALUE>
    <VALUE>BuildingColorSelection</VALUE>
  </FLAG_SET>
  <SELECTION_ATTRIBUTES>
    <SELECTION_POINT type="BuildingTypeSelection">
      <ATTRIBUTE nm="listPrice" mapto="price"/>
      <ATTRIBUTE nm="description" mapto="desc"/>
    </SELECTION_POINT>
    <SELECTION_POINT type="BuildingColorSelection">
      <ATTRIBUTE nm="sku"/>
    </SELECTION_POINT>
  </SELECTION_ATTRIBUTES>
</COMPONENT_DEFINITION

```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126">=>
    TOTAL_PRICE="2120.00" solveDate="20010625">
    <CONFIGURATION>
      <CHOICES>
        <CH DP="BuildingTypeSelection" DM="Apartment" ST="66" QTY="1.0">
          <ATTR NM="price">2000.0</ATTR>
          <ATTR NM="desc">Apartment</ATTR>
        </CH>
        <CH DP="BuildingColorSelection" DM="Blue" ST="66" QTY="1.0">
          <ATTR NM="sku">CB00255</ATTR>
        </CH>
      </CHOICES>
    </CONFIGURATION>
  </CONFIGURABLE_COMPONENT>
</COMPONENTS>

```

Case 7

Including in the REQUEST the FLAG element specifying choices to be returned, the FLAG_SET element specifying the specific choice(s) to be returned, and the FLAG element specifying the type selectionFilterAttribute causes ConfigDetails to return in the RESPONSE the requested choices for a component that have the attribute(s) specified in the selectionFilterAttribute flag.

Details returned are decision point name, domain member name, state, and quantity.

For example:

REQUEST

```

<COMPONENT_DEFINITION component="Building">
  <FLAG type="detailsReturned" value="selections"/>
  <FLAG_SET type="selectionFilterNames">
    <VALUE>BuildingTypeSelection</VALUE>
    <VALUE>BuildingColorSelection</VALUE>
  </FLAG_SET>
  <FLAG type="selectionFilterAttribute" value="listPrice">

```

```

    <VALUE>2000.0</VALUE>
  </FLAG>
</COMPONENT_DEFINITION

```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id=>
"1676995126" TOTAL_PRICE="2120.00" solveDate="20010625">
    <CONFIGURATION>
      <CHOICES>
        <CH DP="BuildingTypeSelection" DM="Apartment" ST="66" QTY="1.0">
          </CH>
        </CHOICES>
      </CONFIGURATION>
    </CONFIGURABLE_COMPONENT>
  </COMPONENTS>

```

Case 8

Including in the REQUEST the FLAG element specifying choices to be returned, the FLAG_SET element specifying the specific choice(s) to be returned, the FLAG element specifying the type selectionFilterAttribute, and the SELECTION_ATTRIBUTES element specifying the attributes to be returned causes ConfigDetails to return in the RESPONSE the requested choices for a component that have the attribute(s) specified in the selectionFilterAttribute flag. Details returned are decision point name, domain member name, state, quantity, and requested attributes. See cases 2 and 7 above.

For example:

REQUEST

```

<COMPONENT_DEFINITION component="Building">
  <FLAG type="detailsReturned" value="selections"/>
  <FLAG type="selectionFilterAttribute" value="onOrder">
    <VALUE>true</VALUE>
  </FLAG>
  <FLAG_SET type="selectionFilterNames">
    <VALUE>BuildingTypeSelection</VALUE>
    <VALUE>BuildingColorSelection</VALUE>
  </FLAG_SET>
  <SELECTION_ATTRIBUTES>
    <SELECTION_POINT type="BuildingTypeSelection">
      <ATTRIBUTE nm="listPrice" mapto="price"/>
      <ATTRIBUTE nm="description" mapto="desc"/>
    </SELECTION_POINT>
    <SELECTION_POINT type="BuildingColorSelection">
      <ATTRIBUTE nm="sku"/>
    </SELECTION_POINT>
  </SELECTION_ATTRIBUTES>
</COMPONENT_DEFINITION>

```

RESPONSE

```

<COMPONENTS>

```

```

<CONFIGURABLE_COMPONENT name="Building-1" component="Building"
  id="1676995126"⇒
TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    <CHOICES>
      <CH DP="BuildingTypeSelection" DM="Apartment" ST="66" QTY="1.0">
        <ATTR NM="price">2000.0</ATTR>
        <ATTR NM="desc">Apartment</ATTR>
      </CH>
      <CH DP="BuildingColorSelection" DM="Blue" ST="66" QTY="1.0">
        <ATTR NM="sku">CB00255</ATTR>
      </CH>
    </CHOICES>
  </CONFIGURATION>
</CONFIGURABLE_COMPONENT>
<COMPONENTS>

```

Choice Violations

The `Flag` child element of the `COMPONENT_DEFINITION` element lets you retrieve component violations for a choice. Note that choices have to be requested in order to return choice conflicts.

Including in the `REQUEST` a `FLAG` element under the `COMPONENT_DEFINITION` element for a component type specifying choices to be returned, and another `FLAG` element specifying choice violations to be returned, causes `ConfigDetails` to return in the `RESPONSE` the choice violations for each of the choices returned for components of the specified type.

For example:

REQUEST

```

<COMPONENTS>
  <FLAG type="componentFilterType" value="Building"/>
  <COMPONENT_DEFINITION component="Building">
    - - -
    <FLAG type="detailsReturned" value="selections"/>
    <FLAG type="detailsReturned" value="selectionConflicts"/>
    - - -

```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126"⇒
TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    <CHOICES>
      <CH DP="BuildingTypeSelection" DM="Apartment" ST="66" QTY="1.0">
        <VIOLATIONS>
          <EXPLANATION>The country and building type are not compatible.
        </EXPLANATION>

```

```

    </VIOLATIONS>
  </CH>

```

Component Violations

The FLAG child element of the COMPONENT_DEFINITION element lets you retrieve component violations for a component.

Including in the REQUEST the FLAG element under the COMPONENT_DEFINITION element for a component type specifying component violations to be returned, causes ConfigDetails to return in the RESPONSE the component violations for components of the specified type.

For example:

REQUEST

```

<COMPONENTS>
  <FLAG type="componentFilterType" value="Building"/>
<COMPONENT_DEFINITION component="Building">
  - - -
  <FLAG type="detailsReturned" value="componentConflicts"/>
  - - -

```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126" =>
    TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    <VIOLATIONS>
      <EXPLANATION>The building height cannot exceed 500 feet.</
    EXPLANATION>
    </VIOLATIONS>

```

Externs

The FLAG and FLAG_SET child elements of the COMPONENT_DEFINITION element let you retrieve extern values for a component.

Case 1

Including in the REQUEST a FLAG element under the COMPONENT_DEFINITION element for a component type specifying externs to be returned, causes ConfigDetails to return in the RESPONSE all externs for components of the specified type.

For example:

REQUEST

```

<COMPONENTS>
  <FLAG type="componentFilterType" value="ZSer"/>
  <COMPONENT_DEFINITION component="ZSer">
    - - -
    <FLAG type="detailsReturned" value="externs"/>
    - - -
  </COMPONENT_DEFINITION>

```

RESPONSE

```

<COMPONENTS>
<CONFIGURABLE_COMPONENT name="ZSeries05" component="ZSER"
  id="1676995126" TOTAL_PRICE="42120.00" solveDate="20040625">
<CONFIGURATION>
  - - -
  <EXTERN_VARS>
  <EV NM="# of doors" TY="INT">
    <VAL>2</VAL>
  </EV>
  <EV NM="Delivery date" TY="DATE">
    <VAL>20041231</VAL>
  </EV>
  <EV NM="PriceLimit" TY="DOUBLE">
    <VAL>40000</VAL>
  </EV>
  <EV NM="Undercoating?" TY="BOOL">
    <VAL>>false</VAL>
  </EV>
  <EV NM="Wheel type" TY="STRING">
    <VAL>ALLOY</VAL>
  </EV>
</EXTERN_VARS>
  - - -
</CONFIGURATION>

```

Case 2

Including in the REQUEST a FLAG element under the COMPONENT_DEFINITION element for a component type specifying externs to be returned, and a FLAG_SET element under the same COMPONENT_DEFINITION element defining the specific extern(s) to be returned, causes ConfigDetails to return in the RESPONSE only the requested extern(s) for components of the specified type.

For example:

REQUEST

```

<COMPONENTS>
  <FLAG type="componentFilterType" value="ZSER"/>
  <COMPONENT_DEFINITION component="ZSER">
    - - -
    <FLAG type="detailsReturned" value="externs"/>
    <FLAG_SET type="externFilterNames">
      <VALUE># of doors</VALUE>

```

```

    </FLAG_SET>
    - - -
</COMPONENT_DEFINITION>

```

RESPONSE

```

<COMPONENTS>
<CONFIGURABLE_COMPONENT name="ZSeries05" component="ZSER"
  id="1676995126"⇒
TOTAL_PRICE="42120.00" solveDate="20040625">
  <CONFIGURATION>
    - - -
    <EXTERN_VARS>
    <EV NM="# of Doors" TY="2">
      <VAL>2</VAL>
    </EV>
  </EXTERN_VARS>
  - - -
</CONFIGURATION>

```

Numeric Values

The FLAG and FLAG_SET child elements of the COMPONENT_DEFINITION element let you retrieve numeric values for a component.

Case 1

Including in the REQUEST a FLAG element under the COMPONENT_DEFINITION element specifying numeric values to be returned for a component type causes ConfigDetails to return in the RESPONSE all numeric values for components of the specified type.

For example:

REQUEST

```

<COMPONENTS>
  <FLAG type="componentFilterType" value="Building"/>
  <COMPONENT_DEFINITION component="Building">
    - - -
    <FLAG type="detailsReturned" value="expressions"/>
    - - -
  </COMPONENT_DEFINITION>

```

RESPONSE

```

<COMPONENTS>
<CONFIGURABLE_COMPONENT name="Building-1" component="Building"
  id="1676995126"⇒
TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    - - -
    <NUMERIC_VALUES>

```

```

    <NUM NM="TotalListPrice" VL="2001.0" TY="FLOAT"/>
    <NUM NM="OccupantsPerFloor" VL="1.3333334" TY="FLOAT"/>
  </NUMERIC_VALUES>
  - - -
</CONFIGURATION>

```

Case 2

Including in the REQUEST a FLAG element under the COMPONENT_DEFINITION element for a component type specifying numeric values to be returned, and a FLAG_SET element under the same COMPONENT_DEFINITION element defining the specific numeric value(s) to be returned, causes ConfigDetails to return in the RESPONSE only the requested numeric value(s) for components of the specified type.

For example:

REQUEST

```

<COMPONENTS>
  <FLAG type="componentFilterType" value="Building"/>
  <COMPONENT_DEFINITION component="Building">
    - - -
    <FLAG type="detailsReturned" value="expressionsterns"/>
    <FLAG_SET type="expressionFilterNames">
      <VALUE>OccupantsPerFloor</VALUE>
    </FLAG_SET>
    - - -
  </COMPONENT_DEFINITION>

```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126">⇒
  TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONFIGURATION>
    - - -
    <NUMERIC_VALUES>
      <NUM NM="OccupantsPerFloor" VL="1.3333334" TY="FLOAT"/>
    </NUMERIC_VALUES>
    - - -
  </CONFIGURATION>

```

External Variables

The EXTERN_VARS element, the child element EV, and the NM attribute of the EV element let you retrieve extern variable values for the configuration.

All Values

Including in the REQUEST the EXTERN_VARS element causes the COP to return in the RESPONSE all EV elements for the configuration. Each EV element is a child element of EXTERN_VARS that has attribute-value pairs for its NM and TP attributes and one or more VAL child elements that contain the value(s) for the extern variable.

For example:

REQUEST

```
<CONFIGURATION MODEL_ID="AutoInsGeneralCRM" MODEL_VERSION="8-8-1">
  <CHOICES>
    <CH DP="LeadClSelection" DM="DEALER" BY="U" SL="1" QTY="1"/>
    <EVCH DP="ClaimCount">
      <VAL>5</VAL>
    </EVCH>
    <EVCH DP="ContinuousYearsEntry">
      <VAL>15</VAL>
    </EVCH>
  </CHOICES>
  <EXTERN_VARS/>
</CONFIGURATION>
```

RESPONSE

```
<CONFIGURATION MODEL_ID="AutoInsGeneralCRM" MODEL_VERSION="8-8-1"
  COMPILE_VERSION="20020927-152727-590">
  <EXTERN_VARS>
    <EV NM="ClaimCount">
      <VAL>5.0</VAL>
    </EV>
    <EV NM="ContinuousYearsEntry">
      <VAL>15.0</VAL>
    </EV>
    <EV NM="GeneralZipCode">
    </EV>
    <EV NM="VehicleCostsEntry">
    </EV>
    <EV NM="DriverRiskFromCoverage">
    </EV>
  </EXTERN_VARS>
</CONFIGURATION>
```

Selected Values

Including in the REQUEST the EXTERN_VARS element, and the child element EV with a valid value for its NM attribute causes the COP to return in the RESPONSE the EXTERN_VARS element, and as a child element, the EV element named in the request with value(s) for its VAL element(s).

For example:

REQUEST

```

<CONFIGURATION MODEL_ID="AutoInsGeneralCRM" MODEL_VERSION="8-8-1">
  <CHOICES>
    <CH DP="LeadClSelection" DM="DEALER" BY="U" SL="1" QTY="1"/>
    <EVCH DP="ClaimCount">
      <VAL>5</VAL>
    </EVCH>
    <EVCH DP="ContinuousYearsEntry">
      <VAL>15</VAL>
      <VAL>25</VAL>
    </EVCH>
  </CHOICES>
  <EXTERN_VARS>
    <EV NM="ContinuousYearsEntry"/>
  </EXTERN_VARS>
</CONFIGURATION>

```

Configuration Attributes

The FLAG and FLAG_SET child elements of the COMPONENT_DEFINITION element let you retrieve configuration attributes for a component.

Case 1

Including in the REQUEST a FLAG element under the COMPONENT_DEFINITION element for a component type specifying configuration attribute values to be returned causes ConfigDetails to return in the RESPONSE all configuration attributes for components of the specified type.

For example:

REQUEST

```

<COMPONENTS>
  <FLAG type="componentFilterType" value="Building"/>
  <COMPONENT_DEFINITION component="Building">
    - - -
    <FLAG type="detailsReturned" value="configAttributes"/>
    - - -
  </COMPONENT_DEFINITION>

```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126"⇒
  TOTAL_PRICE="2120.00" solveDate="20010625">
    <CONFIGURATION>
      - - -
    <CONFIG_ATTRIBUTES>
      <ATTR NM="Customer name">John Smith
    </ATTR>
      <ATTR NM="Phone number">(123)456-7890

```

```

    </ATTR>
  </CONFIG_ATTRIBUTES>
  - - -
</CONFIGURATION>

```

Case 2

Including in the REQUEST a FLAG element under the COMPONENT_DEFINITION element for a component type specifying configuration attributes to be returned and a FLAG_SET element under the same COMPONENT_DEFINITION element defining the specific configuration attribute(s) to be returned causes ConfigDetails to return in the RESPONSE only the requested configuration attribute(s) for components of the specified type.

For example:

REQUEST

```

<COMPONENTS>
  <FLAG type="componentFilterType" value="Building"/>
  <COMPONENT_DEFINITION component="Building">
    - - -
    <FLAG type="detailsReturned" value="externs"/>
    <FLAG_SET type="configAttributesFilterNames">
      <VALUE>CustomerName</VALUE>
    </FLAG_SET>
    - - -
  </COMPONENT_DEFINITION>

```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126">
    TOTAL_PRICE="2120.00" solveDate="20010625">
    <CONFIGURATION>
      - - -
      <CONFIG_ATTRIBUTES>
        <ATTR NM="Customer name">John Smith
        </ATTR>
      </CONFIG_ATTRIBUTES>
      - - -
    </CONFIGURATION>

```

Hierarchical Component Structure

The STRUCTURE and SUBSTRUCTURE child elements of the SECTION element let you return high-level details for components of a compound configuration in a hierarchical structure. The STRUCTURE element always defines a component type filter whereas the SUBSTRUCTURE elements may define either a component type filter or a connection type filter. There can be only one STRUCTURE element per SECTION element. However there can be one or more SUBSTRUCTURE elements under the STRUCTURE element and nested SUBSTRUCTURE elements as well.

Including in the REQUEST the STRUCTURE element defined as

```
<SECTION>
<STRUCTURE value="Building">
<SUBSTRUCTURE type="componentFilterType" value="Floor">
<SUBSTRUCTURE type="connectionFilterType"
value="RoomOnFloor"/>
</SUBSTRUCTURE>
</STRUCTURE>
```

causes ConfigDetails to return in the RESPONSE all components of type Building, all components of type Floor connected to each Building, and all components connected to each Floor by the connection type RoomOnFloor in a hierarchical structure.

For example:

REQUEST

```
<SECTION>
<STRUCTURE value="Building">
  <SUBSTRUCTURE type="componentFilterType" value="Floor">
    <SUBSTRUCTURE type="connectionFilterType" value="RoomOnFloor"/>
  </SUBSTRUCTURE>
</STRUCTURE>
```

RESPONSE

```
<SECTION>
<STRUCTURE>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id=>
"1676995126" TOTAL_PRICE="2120.00" solveDate="20010625">
  <CONNECTED_COMPONENT>
    <CONNECTION id="1676995111" name="BtF" ref="FloorInBuilding"
      fromCompId=>
"1676995126" fromCompName="Building-1" fromCompType=
  "Building" toCompId=>
"1676995128" toCompName="Floor-1" toCompType="Floor"/>
    <CONFIGURABLE_COMPONENT name="Floor-1" component="Floor" id="987654322"
    =>
      TOTAL_PRICE="1900.00" solveDate="20010625">
    <CONNECTED_COMPONENT>
      <CONNECTION id="1676995112" name="FtR" ref="RoomOnFloor" fromCompId=
    =>
      "1676995128" fromCompName="Floor-1" fromCompType="Floor" toCompId=
    =>
      "1676995127" toCompName="Room-1" toCompType="Room"/>
    <CONFIGURABLE_COMPONENT name="Room-1" component="Room" id="987654323"
    =>
      TOTAL_PRICE="1120.00" solveDate="20010625" />
    </CONNECTED_COMPONENT>
  </CONFIGURABLE_COMPONENT>
</CONNECTED_COMPONENT>
</CONFIGURABLE_COMPONENT>
</STRUCTURE>
```

```
</SECTION>
```

Connections

The CONNECTIONS child element of the SECTION element and its FLAG child element let you retrieve connections and their details. There can be only one CONNECTIONS element per SECTION element. The FLAG element of CONNECTIONS lets you define the connection type filter. There may be zero or more FLAG elements, each defining a different connection type filter.

Case 1

Including in the REQUEST only the CONNECTIONS element without defining the FLAG element causes ConfigDetails to return in the RESPONSE all details for connections if connected to components that are returned in the response.

For example:

REQUEST

```
<SECTION nm="PackageDetails">
  <CONNECTIONS>
</CONNECTIONS>
```

RESPONSE

```
<SECTION nm="PackageDetails">
  <CONNECTIONS>
    <CONNECTION id="1676995111" name="BtF" ref="FloorInBuilding" fromCompId=
⇒
    "1676995126" fromCompName="Building-1" fromCompType="Building" toCompId=
⇒
    "1676995128" toCompName="Floor-1" toCompType="Floor"/>
    <CONNECTION id="1676995112" name="FtR" ref="RoomOnFloor" fromCompId=
⇒
    "1676995128" fromCompName="Floor-1" fromCompType="Floor" toCompId=
⇒
    "1676995127" toCompName="Room-1" toCompType="Room"/>
  </CONNECTIONS>
```

Case 2

Including in the REQUEST the CONNECTIONS element and the FLAG child element defining one or more connection type filter(s) causes ConfigDetails to return in the RESPONSE details for connections of the requested connection type(s) if connected to components that are returned in the response.

For example:

REQUEST

```
<SECTION nm="PackageDetails">
  <CONNECTIONS>
    <FLAG type="connectionFilterType" value="RoomOnFloor"/>
  </CONNECTIONS>
```

RESPONSE

```

<SECTION nm="PackageDetails">
  <CONNECTIONS>
    <CONNECTION id="1676995112" name="FtR" ref="RoomOnFloor"
      fromCompId="1676995128" fromCompName="Floor-1" fromCompType="Floor"
      toCompId=>
    =>
    =>
    =>
    =>
    =>
    "1676995127" toCompName="Room-1" toCompType="Room"/>
  </CONNECTIONS>

```

Completeness Information

Completeness information will be returned in the response for those components for which component conflicts have been requested and if the attribute *validate* of the CONFIGURATION element is set to “true” in the request. “The CONFIGURATION element” and “ComponentConflicts”

See [Chapter 17, “Retrieving Saved Configuration Information,” The CONFIGURATION Element, page 231.](#)

Including in the REQUEST

```

<CONFIGURATION configId="1676995129" solutionId="OutputUtil"
  validate="true">

```

and

```

<COMPONENTS>
  <FLAG type="componentFilterType" value="Building"/>
  <COMPONENT_DEFINITION component="Building">
    - - -
  <FLAG type="detailsReturned" value="componentConflicts"/>
  - - -

```

causes ConfigDetails to return in the RESPONSE a status of FALSE along with the required decision point name(s) that do not have a selection if components of type Building have a completeness violation, OR, a status of TRUE if the components of type Building do not have any completeness violations.

For example:

REQUEST

```

<CONFIGURATION configId="1676995129" solutionId="OutputUtil" validate="true">
  and
  <COMPONENTS>
    <FLAG type="componentFilterType" value="Building"/>
    <COMPONENT_DEFINITION component="Building">
      - - -
    <FLAG type="detailsReturned" value="componentConflicts"/>
    - - -

```

RESPONSE

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126" =>
    TOTAL_PRICE="2120.00" solveDate="20010625">
    <CONFIGURATION>
      <COMPLETE STATUS="FALSE">
        <DP NM="BuildingHeightSelection"/>
        <DP NM="BuildingStyleSelection"/>
      </COMPLETE>

```

OR

```

<COMPONENTS>
  <CONFIGURABLE_COMPONENT name="Building-1" component="Building"
    id="1676995126" =>
    TOTAL_PRICE="2120.00" solveDate="20010625">
    <CONFIGURATION>
      <COMPLETE STATUS="TRUE">
    </COMPLETE>

```

Summary of Configuration Information Elements and Attributes

These tables provide a quick reference to configuration information elements and attributes.

Element CONFIGURATION		
Attribute	Possible Values	Case-Sensitive?
validate	true	no
	false	no

Element DELTA_INFO – child of element CONFIG_DETAILS		
Attribute	Possible Values	Case-Sensitive?
returnDeltas	true	no
	false	no
startDate	valid Java date	no
endDate	valid Java date	no

Element FLAG – child of element CONFIG_DETAILS				
Attribute	Possible Values	Attribute	Possible Values	Case-Sensitive?
type	detailsReturned	value	totalPrice	no
			compoundConflicts	no

Element SECTION – child of Element CONFIG_DETAILS		
Attribute	Possible Values	Case-Sensitive?
nm	Any string	no

Element FLAG – child of element COMPONENTS				
Attribute	Possible Values	Attribute	Possible Values	Case-Sensitive?
type	componentFilterType	value	Any component type	yes

Element FLAG – child of element CONNECTIONS				
Attribute	Possible Values	Attribute	Possible Values	Case-Sensitive?
type	connectionFilterType	value	Any connection type	yes

Element COMPONENT_DEFINITION – child of element COMPONENTS		
Attribute	Possible Values	Case-Sensitive?
component	Any component type	yes

Element STRUCTURE – child of element SECTION		
Attribute	Possible Values	Case-Sensitive?
value	Any component type	yes

Element SUBSTRUCTURE – child of elements STRUCTURE or SUBSTRUCTURE				
Attribute	Possible Values	Attribute	Possible Values	Case-Sensitive?
type	componentFilterType	value	Any component type	no
	connectionFilterType		Any connection type	no

Element FLAG – child of element COMPONENT_DEFINITION				
Attribute	Possible Values	Child/Attribute	Possible Values	Case-Sensitive?
type	detailsReturned	value	selections	no
	detailsReturned		externs	no
	detailsReturned		expressions	no

Element FLAG – child of element COMPONENT_DEFINITION				
Attribute	Possible Values	Child/Attribute	Possible Values	Case-Sensitive?
	detailsReturned		configAttributes	no
	detailsReturned		selectionConflicts	no
	detailsReturned		componentConflicts	no
type	selectionFilterAttribute	value	Any attribute name	yes
		value	The value for the attribute	yes

Element FLAG_SET – child of element COMPONENT_DEFINITION				
Attribute	Possible Values	Child	Possible Values	Case-Sensitive?
type	selectionFilterNames	VALUE	Any choice name	yes
type	externFilterNames	VALUE	Any extern name	yes
type	expressionFilterNames	VALUE	Any numeric value name	yes
type	configAttributeFilterNames	VALUE	Any config attribute name	yes

Element SELECTION_POINT – child of element SELECTION_ATTRIBUTES		
Attribute	Possible Values	Case-Sensitive?
type	Any choice name	yes

Element ATTRIBUTE – child of element SELECTION_POINT		
Attribute	Possible Values	Case-Sensitive?
nm	Any attribute name	yes
mapTo	Any string	yes

CHAPTER 18

Copying a Configuration

The Configurator XML interface lets you copy and save a previously saved configuration. The copy request is processed separately from, and supersedes any other elements included in the same request. Therefore, if the `configCopy` attribute is included in the request and its value is true, then the only operation performed during that post to the servlet will be the configuration copy. Any other elements/attributes in the request will be ignored.

Elements and Attributes

The COP XML request must include these attributes to copy a configuration.

```
CONFIGURATION configId solutionId configCopy copyName
```

The COP XML response may include these elements and attributes to return information about a copied configuration.

```
ConfigCopy configId solutionId configCopy copyName  
Copy status configId
```

Attributes are:

configId	Specifies the id of the configuration that you want to copy.
solutionId	Indicates the solution for the configuration. (A solution is the implementation of the PeopleSoft Configurator application.)
configCopy	Indicates that this is a copy operation.
copyName	Indicates the name to give to the new copy of the configuration.

The following is a sample copy request:

```
<CONFIGURATION configId="1438491808" solutionId="TelcoDemo"  
  configCopy="true"=>  
  copyName="New Compound"/>
```

Copy and Response

The Copy element and its attributes *status* and *configId* are returned in the response from a `configCopy` request. The attribute *status* indicates the results of the copy operation. The following lists valid status codes and descriptions:

0	Copy was successful.
---	----------------------

- 101** Invalid solution ID.
- 102** Invalid Config ID.
- 103** Solution doesn't allow new configurations.
- 104** Database error on save.

The attribute *configId* is the configId of the new copy of the configuration if the copy was successful, otherwise it is 0.

The following is a sample response for a *SUCCESSFUL* copy:

```
<ConfigCopy solutionId="TelcoDemo" configId="1438491808"
  copyName="New⇒
  Compound"><Copy status="0"
  configId="932435623"/>
</ConfigCopy>
```

The following is a sample response for an *UNSUCCESSFUL* copy:

```
<ConfigCopy solutionId="TelcoDemo" configId="1438491808"
  copyName="New Compound"><Copy status="102" configId="0"/>
</ConfigCopy>
```

CHAPTER 19

Using Batch Configuration Mode

The Configurator XML interface lets you both configure and save configurations in batch mode.

The batch request is processed separately from other elements in the request. In addition, it supersedes any other elements included in the same request. Therefore, if the batch attribute is included in the request and its value is true, then the only operation performed during that post to the servlet will be the batch configuration processing. Any other elements/attributes in the request will be ignored.

Elements and Attributes

The COP XML request must include these attributes to process a configuration in batch mode.

```
CONFIGURATION batch
CONFIG_XML
```

The batch attribute indicates whether this is a batch request. Valid values are true/false.

The COP XML response may include these elements and attributes to return information about a batch configuration.

```
CONFIG_XML isValid wasSaved
VIOLATIONS
  EXPLANATION
Component/CompoundConfiguration
```

isValid indicates if the configuration was valid. Values are true/false.

The method *wasSaved* indicates whether the configuration was saved. Values are true/false.

The VIOLATIONS element contains an EXPLANATION element for every violation in the configuration. Each EXPLANATION element contains the why help for that particular violation.

See Also

[Chapter 19, “Using Batch Configuration Mode,” Configuring a Component, page 265](#)

[Chapter 19, “Using Batch Configuration Mode,” Configuring a Compound Configuration, page 266](#)

Configuring a Component

To configure a component configuration in batch mode, the CONFIG_XML element in the request may include these elements and attributes to contain the information about the configuration:

```

Component name comment solutionId id
CONFIGURATION COMPILE_VERSION SOLVE_DATE MODEL_ID MODEL_VERSION
ATTRIBUTE_SET
  ATTRIBUTE name
  CH DM DP QTY TY
  EVCH DP TY VAL
  NUM NM TP VAL

```

The COP XML response may include these elements and attributes to return information about a component configuration.

```

Component name comment solutionId id configCode
CONFIGURATION COMPILE_VERSION XML_GENERATED_DATE SOLVE_DATE MODEL_ID MODEL_VERSION
ATTRIBUTE_SET
  ATTRIBUTE name
  CH DM DP QTY TY
  EVCH DP TY VAL
  NUM NM TP VAL

```

Note that the request and response are essentially identical and are the product of a call to the `calico.configurator.cop.Component.toXML` method.

Note. The ST attribute is not used.

Configuring a Compound Configuration

To configure a compound configuration in batch mode the `CONFIG_XML` element in the request may include these elements and attributes to contain the information about the configuration:

```

CompoundConfiguration name type version owner comment solutionId id
Components
  ConfigurableComponent name component comment id violation
  CONFIGURATION COMPILE_VERSION SOLVE_DATE MODEL_ID MODEL_VERSION
  ATTRIBUTE_SET
  ATTRIBUTE name
  CH DM DP QTY TY
  EVCH DP TY VAL
  NUM NM TP VAL
Connections
  Connection id name ref comment component ReverseConnection
Structure
  ConnectedComponent id component name

```

The COP XML response may include these elements and attributes to return information about a compound configuration.

```

CompoundConfiguration name type version owner comment solutionId id configCode
Components
  ConfigurableComponent name component comment id violation
  CONFIGURATION COMPILE_VERSION XML_GENERATED_DATE SOLVE_DATE MODEL_ID MODEL_⇒
VERSION
  ATTRIBUTE_SET
  ATTRIBUTE name

```

```

    CH DM DP QTY TY
    EVCH DP TY VAL
    NUM NM TP VAL
Connections
    Connection id name ref comment component ReverseConnection
Structure
    ConnectedComponent id component name

```

Note that the request and response are virtually identical and are the product of a call to the method `calico.cms.runtime.CompoundConfiguration.toXML`.

Note. The ST attribute is not used.

Note. When creating a compound configuration using the COPXML servlet batch mode for a compound with multiple components, only the last component described in the XML message is created. Unless you include the same ID twice in the XML, Advanced Configurator sees both objects as the same thing (and therefore, the second element with the ID 0 overwrites the first).

Saving a Configuration

The COPXML request to save a component/compound configuration in batch mode is the same as the request to configure with the addition of a `saveConfig` attribute on the CONFIGURATION element.

Valid values for the `saveConfig` attribute are:

ALWAYS	Always saves the configuration.
VALID	Only saves the configuration if it is valid.

The COPXML response from a request to save a component/compound configuration in batch mode contains the same structure as the response to a configure request.

Retrieving a Configuration

The COPXML request must include these attributes to retrieve the XML for a configuration in batch mode.

```
CONFIGURATION configId solutionId configXml validate
```

Attributes are:

configId	Specifies the id of the configuration that you want to retrieve.
solutionId	Indicates the solution for the configuration.
configXml	Indicates that this is a retrieve operation.
validate	Indicates to validate the configuration before generating the response.

The COPXML response contains the same structure as the response to a configure request. Please refer to appropriate configure section for details.

CHAPTER 20

Changing the Order Status of a Configuration

The Configurator XML interface lets you change the order status of a previously saved configuration.

The order change request is processed separately from other elements in the request. In addition, it supersedes any other elements included in the same request. Therefore, if the `orderChange` attribute is included in the request and its value is true, then the only operation performed during that post to the servlet will be the order change. Any other elements/attributes in the request are ignored. This chapter describes how to change the order status of a configuration using COPXML.

Elements and Attributes

The COP XML request must include these attributes to change the order status of a configuration.

```
CONFIGURATION configId solutionId orderChange newState
```

The COP XML response may include these elements and attributes to return information about an order change request.

```
OrderChange configId solutionId newState stateChanged
```

Attributes are:

configId	Specifies the id of the configuration for which you want to change the status.
solutionId	Indicates the solution for the configuration.
orderChange	Indicates that this is a order change operation.
newState	Indicates the new state value for the order status. Valid values are <i>Save</i> , <i>Submit</i> , <i>Cancel</i> , and <i>Delete</i> .

Example order change request:

```
<CONFIGURATION configId="1004316094" solutionId="BMWTest"
  orderChange="true" new⇒
  State="Submit"/>
```

Order Change and Response

The `OrderChange` element and its attributes `configId`, `solutionId`, `newState`, and `stateChanged` are returned in the response from an orderChange request.

The attribute `stateChanged` indicates the results of the order change operation. The attribute `newState` indicates the new order status value.

PART 5

PeopleSoft CRM Order Capture Integration

Chapter 21

Understanding Integration with PeopleSoft CRM Order Capture

Chapter 22

Setting Up Integration

CHAPTER 21

Understanding Integration with PeopleSoft CRM Order Capture

This chapter discusses:

- Integration with PeopleSoft Enterprise Order Capture applications.
- Security.

Integration with PeopleSoft Enterprise Order Capture Applications

A customer or service agent using a PeopleSoft CRM Order Capture application can sign in to PeopleSoft Order Capture, create a new order or open an existing one, click the Configurator icon on the Order page, display the product configuration page, make changes to the order, save it, and pass the configuration and order data to the CRM system for storage or further processing.

Users can perform these operations from PeopleSoft Enterprise Order Capture and Order Capture Self Service. Integrating Advanced Configurator enhances:

- Quote and order processes.
- Pricing.
- Installed product configuration and service maintenance.

When integrated with Order Capture applications, Advanced Configurator:

- Ensures that the product selections are compatible and correct.
- (Optionally) provides the list price of the configured product.
This price may be further surcharged or discounted by the Enterprise Pricer engine.
- Calculates and displays delta pricing, in which users can observe the effects of their selections on pricing.
- Displays the details of a configuration within Order Capture.
- After the configuration session, returns the user to the calling component (such as Order Capture).

Insurance and Financial Products

Agents can use PeopleSoft Enterprise Order Capture to take applications for financial services and process them. If the services are configurable, the configuration user interface (UI) appears for selections to be made. When the session is complete, the agent returns to the order form and continues. Customers can perform these actions through Order Capture Self Service.

Similarly, insurance products such as coverages and deductibles are more efficiently ordered and maintained when you use the integrated Advanced Configurator.

Service Products

Users can quote and order services through Order Capture. If the services are configurable, the user can access the configuration page from the order or quote in the same way as any other configurable product.

Security

Advanced Configurator can optionally utilize the user sign-in and authentication that is provided when you sign in to the PeopleSoft Enterprise Order Capture application. Invoking the configuration UI from an Order page requires no additional sign-in.

CHAPTER 22

Setting Up Integration

This chapter discusses how to:

- Set up PeopleSoft Advanced Configurator for integration.
- Set up PeopleSoft Enterprise CRM to integrate with Advanced Configurator.
- Create Advanced Configurator schemas.
- Access the Advanced Configurator Solution from Within PeopleSoft CRM.

Setting Up PeopleSoft Advanced Configurator for Integration

Perform the following steps to set up your PeopleSoft Advanced Configurator integration with PeopleSoft CRM.

1. Install PeopleSoft Advanced Configurator Server and deploy the desired solutions on this Configurator Server.

See *PeopleSoft Enterprise CRM 9 Installation Guide*

2. Navigate to PeopleTools, Integration Broker, Gateways, and click the Search button on the Gateways search page.

This accesses the Gateway ID: LOCAL page.

3. Enter the Gateway Uniform Resource Locator (URL) as `http://<< PeopleSoft Web Server >>/PSIGW/PeopleSoftListeningConnector`, and click Save.

Note. Remember that the URL is case-sensitive.

4. Load the Connector information by clicking the Load button.

A Loading Process was successful message appears.

5. Click OK to continue.

6. A grid appears.

This grid displays all of the loaded connectors. For each connector ID, a connector class name exists.

7. Click Save.

8. Click the Refresh button next to Refresh Integration Gateway.properties file.

A Gateway Refresh Process was successful message appears.

9. Click OK to continue, then Save.

10. Access the Node Definitions page from PeopleTools, Integration Broker, Node Definitions.

11. Open node PSFT_CFG.
12. On the Connectors tab for the new node name, (where the value of Gateway ID should be LOCAL and Connector ID should be HTTPTARGET), change the “PRIMARYURL” property to be the URL of your Advanced Configurator server. (The URL is case-sensitive.)

Note. Advanced Configurator integration does not support URLs beginning with *https* for use with Secure Socket Layers (SSL).

13. Save these settings.
14. Enter PSFT_CFG in the Message Node Name field.

You may want to set up two nodes: one for internally facing applications and one for Self Service (external) applications. If so, you can create another node that is identical to the shipped PSFT_CFG node in every respect other than its name and the associated URL
15. Navigate to the Installations page (Set Up CRM, Product Related, Advanced Configurator, Installation) and identify an internal node, an external node, or both.

The internal node is used for all internally facing applications. The external node is the one that is used for all Self Service applications (see below).
16. Navigate to the Schemas page (Set Up CRM, Product Related, Advanced Configurator, Schemas) to create schemas and link them to solutions on the PeopleSoft Advanced Configurator Server.

See [Chapter 22, “Setting Up Integration,” Creating Advanced Configurator Schemas, page 278](#).
17. When you set up items, products, or packages in the PeopleSoft CRM Item Definition and Product Definition components, be sure to link them to the correct PeopleSoft Advanced Configurator schemas that were established in step 5.

See *PeopleSoft Enterprise CRM 9 Product and Item Management PeopleBook*, “Defining Items” and *PeopleSoft Enterprise CRM 9 Product and Item Management PeopleBook*, “Setting Up Products”.

Setting Up PeopleSoft CRM to Integrate with Advanced Configurator

This section provides an overview of integration setup features and explains how to use PeopleSoft CRM setup pages to activate the integration between PeopleSoft Advanced Configurator and PeopleSoft CRM applications. Specifically, it explains how to associate configuration messaging nodes with the CRM Application and enable Advanced Configurator debugging

To associate Advanced Configurator Messaging Node and enable debugging, use the Installation (CFG_SETUP) component.

Page Used to Set Up Configurator Integration with PeopleSoft CRM

Page Name	Object Name	Navigation	Usage
Installation Table	CFG_SETUP	Set Up CRM, Product Related, Advanced Configurator, Installation, Installation Table	Use the Installation Table page to associate configuration messaging nodes with CRM Applications and enable Configurator debugging

Associating Advanced Configurator Messaging Node and Enabling Debugging

Access the Installation Table page.

The screenshot shows the 'Installation Table' page. It has two main sections: 'Integration Broker Setup' and 'Configurator Debug Information'. In the 'Integration Broker Setup' section, there is a dropdown menu for '*Configurator Server Node' set to 'Define Internal/External Nodes'. Below it are two text input fields: 'Internal Node' and 'External Node', both containing the value 'PSFT_CFG'. In the 'Configurator Debug Information' section, there is a 'Debug' dropdown menu set to 'Off'.

Installation Table page

The Installation Table page enables you to specify the PeopleSoft Enterprise Integration Broker Messaging Node for internal CRM applications, such as Order Capture. You also can specify an external node for customer-facing applications such as Order Capture Self Service. Be sure to specify at least one node; otherwise, an error message is generated. Order Capture and Order Capture Self Service use these nodes to call the Advanced Configurator server at run time.

Integration Broker Setup

Configurator Server Node Select *Define External Node Only*, *Define Internal Node Only*, or *Define Internal/External Nodes*, depending on whether you want to use Advanced Configurator with internal applications, external self-service CRM applications, or both. When using both internal and external CRM applications, you can define a separate node for the self-service application so that transactions are stored on a separate, secure server.

Note. The Internal Node and External Node fields appear or disappear depending on your selection.

Internal Node Enter a configurator node name. This node is used to integrate the configurator server with internal-facing CRM applications, such as PeopleSoft Order Capture.

Note. Nodes are available from the drop-down list, which prompts against the PSMMSGNODEDEFN table.

External Node Enter a configurator node name. This node is used to integrate the configurator server with customer-facing CRM applications, such as Order Capture Self Service.

Configurator Debug Information

Debug When you turn debugging on, raw configuration details are displayed in XML format at runtime. Specifically, this means that when you have completed your configuration session, you are presented with two pages prior to returning to the application that invoked PeopleSoft Advanced Configurator. These two pages contain an XML request and an XML response. The XML request page displays the XML data that was sent to the configurator server to retrieve information about the configuration; the XML response page displays the XML data that was returned by the configurator server. This is a useful tool for debugging your schemas because you can quickly verify that XML configuration details are being returned from the calling application just as you would like them to be. When you turn debugging off (default), you do not see the two XML pages prior to returning to the calling application from your configuration session.

The Configurator Solution Tester also enables this request and response XML to be displayed, as well as providing further debugging options.

Note. Log files are available from the appropriate directory on the configurator server when Configurator Debug is activated. For example, if your Advanced Configurator server is running on Microsoft Windows, the logs are stored in C:\bea\weblogic81\config\CalicoDomain\logs

Creating Advanced Configurator Schemas

This section provides an overview of configurator schemas and discusses how to:

- Create schemas for external solutions.
- Create schemas for internal solutions.
- Establish configuration display and pricing options.
- Specify request details.

To create schemas for external solutions, use the Schema (CFG_SCHEMA_GBL) component.

Pages Used to Create Advanced Configurator Schemas

Page Name	Object Name	Navigation	Usage
Display	CFG_SCHEMA_DISPLAY	Set Up CRM, Product Related, Advanced Configurator, Schemas, Schema Setup Set Up CRM, Product Related, Advanced Configurator, Schemas, Display Select Internal Solution.	Create schemas for external solutions. Create schemas for internal solutions.
Price	CFG_SCHEMA_PRICE	Set Up CRM, Product Related, Advanced Configurator, Schemas, Price	Use the Price page to specify options to return the configuration list price, as well as to establish recurring pricing for configured products.
Request Details	CFG_SCHEMA_OUTPUT	Set Up CRM, Product Related, Advanced Configurator, Request Details	Use the Request Details page to select the configuration information that you want the user to receive.

Understanding Configurator Schemas

Configuration schemas establish the display, pricing, and configuration details for specific a configuration and what information to retrieve from the configuration models on the configurator server. Three schema setup pages are available for Advanced Configurator:

Display	Schemas	Define the Configuration User Interface to be displayed at run time.
Price	Schemas	Define the configuration pricing options.
Request Details	Schemas	Define the configuration details to be sent to and received from the configurator server.

External Solutions and Internal Solutions

Two types of configurator schemas are available: externally created solutions and internally defined solutions. External and internal solutions can be distinguished like this:

External Solution

Enables you to direct runtime data from the model to an HTML-based user interface (UI) that is built specifically for this solution with Java Server Pages (JSP) and external HTML editing tools (such as Macromedia Dreamweaver).

Note. Templates for Micromedia Dreamweaver are bundled with the PeopleSoft Advanced Configurator application.

Internal Solution Enables you to define the user interface from within the PeopleSoft CRM schema setup pages.

Creating Schemas for External Solutions

Access the Display page. Select *External Solution* as the solution type.

The screenshot shows a web form with the following fields and values:

- Schema ID:** DSL_COMPUTER
- *Description:** DSL Computer
- Solution Properties:**
 - Frame Dimensions:** 800 x 1200
 - *Solution Type:** External Solution
 - External Solution:** DSLComputer
- Buttons:** Solution Tester, Model Tester

Display page for external solution

Schema ID The schema ID is a unique identifier for the configuration schema. The schema ID is the identifier that is associated with an item or product so that the system knows how to properly configure the product or package.

Note. The schema ID should match the solution ID if you are using an external solution.

Description Enter a description for the schema ID.

Solution Properties

Frame Dimensions Specify the width and height of the embedded, runtime configuration page in pixels.

Note. When you save the page, validation logic ensures that neither page width nor height are fewer than 800 pixels.

Solution Type Select *External Solution* to select from the existing (externally defined) sets of configurator solutions on the configurator server.

External Solution When the solution type is *External*, you use a predetermined list of solutions, each of which already has the user interface display set. Click the Lookup button to select an external (predefined) solution from a list of all the current configurator solutions that exist on the configurator server.

Solution Tester Select to launch the Configurator Solution Test tool, which displays the user interface for this solution ID.

Model Tester link Select to test model functionality for the underlying model of this solution ID.

Creating Schemas for Internal Solutions

Access the Display page.

Select *Internal Solution* as your solution type.

Note. This page appears when you select a solution type of *Internal Solution*. This page enables you to specify your own solution type instead of choosing one from the default list that is available when you select an external solution. Note that internal configurations can only be used when the configuration server is using the PeopleSoft Enterprise CRM database and not when it is using its own standalone database.

The screenshot shows a web interface for configuring an internal solution. At the top, there are three tabs: 'Display', 'Price', and 'Request Details', with 'Display' being the active tab. Below the tabs, the 'Schema ID' is set to 'DSL_COMPUTER'. The '*Description' field contains 'DSL Computer'. A section titled 'Solution Properties' contains several fields: 'Frame Dimensions' with values '800' and '1200', '*Solution Type' set to 'Internal Solution', '*Configuration Type' set to 'Component', and 'Model ID' set to 'DSLComputer'. There are also links for 'Solution Tester' and 'Model Tester'. At the bottom of this section, the checkbox 'Use Most Current Version' is checked.

Display page for internal solution (1 of 3)

Schema ID The schema ID is a unique identifier for the configuration schema.

Description Enter a unique description for the schema ID.

Solution Properties

Frame Dimensions Specify the width and height of the runtime configuration page in pixels.

Note. When you save the page, validation logic ensures that neither page width or height are fewer than 800 pixels.

Solution Type Select *Internal Solution*, which enables you to define your own solution without returning to the environment of the PeopleSoft Advanced Configurator.

Note. When you select *Internal Solution*, the page is updated to display additional sections, which are explained subsequently.

Configuration Type Select *Component* when the solution contains a single model. Select *Compound* if the solution contains multiple models. If you select *Component*, the list displays only the solutions on the server that contain a single component model. Likewise, selecting *Compound* displays a list of the solutions on the server that contain multiple models.

Note. If you select *Component*, all fields on the Display page will be available for updating.

If you select *Compound*, the only modifiable fields will be the Compound ID, Model Version, Page Title, Validation on Return, and Captions fields. You need to have the individual component schemas already defined.

Model ID or Compound ID Select a model ID (or, in the case of a compound model, a compound ID) for the internal solution.

Use Most Current Version Select to use the most current model.

Note. When you select this check box, the Model Version options disappear.

Solution Tester Click to launch the Configurator Solution Test Tool, which launches the model, any database connections, and the actual user interface. Use this tool to verify that the results from the business logic are properly displayed and to check and tune presentation layout (if it is a custom UI) and navigation. This link is keyed by solution ID.

Model Tester Click to test constraints and conditions that are defined in the model. A test UI is launched, so you can divide testing into two phases: business logic (model constraints and calculations) and presentation (UI, using the Solution Tester). This link is keyed by solution ID.

Display page for internal solution (2 of 3)

Display Properties

Page Title Enter a title for the configuration display.

Restore Policy You can select a restore policy that selects either the original model version or the most current model version. This option gives you control over whether the user's older, saved configurations are run against the newest model version if the user requests a saved configuration to view or to use as the basis for a new configuration. The problem to consider is whether the new model, which can change substantially through updates, can properly display and process configuration data that is produced by an older model.

Page Information

Tab You can control the number of tabs that appear on the page. Enter the number of the tab here, and make your selections for the content of the tab in the Control Setup section.

Tab Caption Define a label for each tab.

Number of Columns Define the number of column controls that you want to appear in the configuration page for this tab.

Control Setup

Sequence Determine the sequence of the controls that are displayed on the tabs that you create for the configuration page.

Type Select *Configuration Attribute*, *Expression*, *External Variable*, or *Selection*.

Name Enter a name for the configuration attribute, expression, external variable, or selection.

Attribute Define the attribute that appears in the drop-down list or as a radio button selection.

Caption Enter a descriptive caption that appears on the page.

Control Type Select to render the control type as either a drop-down list, or radio button..

Field Processing Select whether this runtime page uses dynamic or deferred processing. The *Dynamic* option causes the page to refresh automatically when the user enters data and presses the Tab key to move out of a field. The *Deferred* option leaves processing until the user clicks the Submit button.

Control Size Specify the maximum number of characters to allow in the control. This option appears if the control type is Configuration Attribute or External Variable.

Display Options

These options appear only for control type Selection.

Show Violations Select to display red violations text on the selections.

Show Eliminated Select to show options that are constrained away by previous selections. If this check box is cleared, invalid selections will not appear at all.

Show Delta Price If you have pricing information, select this check box to display pricing change in the form of how much has been added to or subtracted from the price. This is also known as the ability to show *plus-minus pricing*.

Display page for internal solution (3 of 3)

Captions

Return, Return to Manager, Cancel, Update, and None

Specify the text to use on the labels of the action buttons. These buttons will be viewable on the configuration page. Return, Cancel, Update, and None are available for component models. Return, Cancel, and Return to Manager are available for compound models.

Miscellaneous

Validate on Return

Select to check the validity of the configuration before returning to the calling CRM application.

Show Configuration List Price

Select to display the list price at the top of the configuration page.

Show Application Violations

Select to show violation messages during product configuration. These violation messages appear at the top of the configuration page in a red text and red button format.

Establishing Configuration Display and Pricing Options

Access the Price page.

Display Price Request Details

Schema ID WALKIN_FREEZER

*Description Walk-in Freezer

Price Mode

Mode: Standard / Advanced

Pricing Properties

*List Price Source Configurator

Configuration List Price

Type	Name	Attribute	Operator	Type	Name	Delta Price Only
Selection	CoolingUnitsSelection	LIST_PRICE	*	Expressio	fxCurrencyFa	<input type="checkbox"/>
Selection	DoorsFrontSelection	LIST_PRICE	*	Expressio	fxCurrencyFa	<input type="checkbox"/>
Selection	DoorsLeftSelection	LIST_PRICE	*	Expressio	fxCurrencyFa	<input type="checkbox"/>
Selection	DoorsRearSelection	LIST_PRICE	*	Expressio	fxCurrencyFa	<input type="checkbox"/>
Selection	DoorsRightSelection	LIST_PRICE	*	Expressio	fxCurrencyFa	<input type="checkbox"/>

*Recurring Price Source Product Definition

Price page (1 of 2)

Pricing Properties

*List Price Source Product Definition

*Recurring Price Source Configurator

Recurring Price

Recurring	Expression
Recurring Price	<input type="text"/>
Recurring Frequency	<input type="text"/>
Frequency Description	<input type="text"/>

Price page (2 of 2)

Price Mode

The Price Mode group box appears when you specify *Configurator* in the List Price Source option.

Mode

Indicates which price mode, Standard or Advanced, is employed to define the pricing schema as shown on the Configuration List Price grid. Clicking the *Advanced* link makes operators and expressions available to further define each control's pricing.

Click Standard to change the mode from Advanced and remove the operators and expressions. See the following explanation.

Pricing Properties

List Price Source

Select *Configurator* or *Product Definition* to indicate whether to draw pricing data for the controls from the configurator model or from the product definition.

Selecting *Configurator* displays the Configurator List Price grid, in which you specify the controls to be priced and their definitions. The product definition contains this information already.

Configuration List Price

The configuration list price is a list price. PeopleSoft Enterprise Pricer can act further on this price and may place a surcharge on it or discount it, depending on the setup in the Enterprise Pricer application. However, you can operate on these values using the Operator drop-down described subsequently. In addition, the values that are represented on each of the rows in the grid are summed to yield the total price.

This section is available only when you have selected *Configurator* in the List Price Source field.

Type Select *Expression* to identify and define an expression from the configurator model to be used to deliver a list price. Select *Selection* to indicate that an attribute of the specified selection is the source for a list price for that selection.

Note. When the Type is Expression, Name is the only field available for editing. The Delta Price Only check box remains available for selection.

Name Type or select a name for the selection or expression from the Name lookup list. Names are supplied by the model.

Attribute Select the attribute of the specified selection from which to take the list price value.

Operator Appears when Price Mode is set to Advanced. Use the Operator drop-down list to define an expression to operate on the list price that is passed to it from the left-hand side of the row.

Delta Price Only Select to return delta price information for the selection or expression. Delta information is a price value that indicates the difference, plus or minus, that the picking of a particular selection had on a price. Delta pricing must be enabled in the model as well.

Recurring Price Source Select *Product Definition* to indicate that recurring pricing information is to be taken from the product definition. Select *Configurator* to define the source for recurring pricing for the product. The Recurring Price grid appears when you select the *Configurator* option.

Recurring Price

Recurring and Expression Select expressions, defined in the configurator model, for:

- **Recurring Price:** Select an expression that provides a value for a recurring charge to be added to the list price.
- **Recurring Frequency:** Select an expression that defines which recurring frequency to use when you add the recurring charge order line, for example, MNTLY.
- **Frequency Description:** Select text that describes the recurring frequency in words, such as Monthly.

Specifying Request Details

The Request Details page enables you to specify in detail what the request will look like that goes from Order Capture to the Advanced Configurator server and in turn, what details on the configuration will be returned to Order Capture from the Advanced Configurator server (all by way of XML). Finally, it enables you to specify what XSLT StyleSheet to use when rendering the configuration details to the user in HTML on the line details page of Order Capture.

Access the Request Details page.

Request Details page (1 of 2)

Name	Attribute
Additional Features	ShortName
Anytime Minutes	ShortName
Mobile-to-Mobile Minutes	ShortName
Weekend Minutes	ShortName

Request Details page (2 of 2)

You should recognize that the Request Details page is affected by the Configuration Type option on the Display page, but is not affected by Solution Type. The option to include components, connections, and structure in the configuration details is not available for the configuration type of Component. However, all options on the Request Details page are available to the user when the configuration type is compound.

Request Details

Request Message	Specify a <i>Custom XML</i> or <i>Default XML</i> output. Configuration details are in XML, and a default XSLT is provided with Configurator. Select <i>Custom XML</i> to access a text entry field where you can define a request.
Define Request	When you select <i>Custom XML</i> , the Define Request link appears. Click the link to access a text entry field into which you can enter the XML request.
Render With	Determines how the information that is returned by the request is displayed. Enter <i>Custom Stylesheet</i> or <i>Default Stylesheet</i> stylesheet. When you select the <i>Custom</i> option, the <i>Define Stylesheet</i> link appears and you can define your own stylesheet. By using a custom stylesheet, you can change the order in which the information is displayed or the amount of information displayed. For example, a custom stylesheet could be used to display expressions first, followed by selection points, and then conflicts. Or you could display just the selection points with selections and not bother to display the actual domain members that are selected or their quantity.
Define Stylesheet	Click to define a custom XSLT stylesheet for this schema ID. The link accesses a page containing a large text entry field into which you can insert a text defining a stylesheet.

Request Properties

Configuration Details	Select this check box to request configuration details that are provided by Configurator. When you select this box, a Configuration Details group box (described in the next section) appears and enables you to make detailed selections. The Configuration Details options determine what is included in part of the default xml request.
Package Components	Select to choose name and attribute options for package components. A Package Components grid appears at the bottom of the page. A package component is one of many products that will eventually make up a package. For example, a computer product may actually consist of several products such as a monitor, keyboard, and mouse, as well as the actual computer.
Product Selector	Select to choose name and attribute options for product selections. A Product Selector grid appears at the bottom of the page. The option replaces the product ID on the order capture line with the product ID that is specified in the chosen selection point.
Purchased Components	Select to choose name and attribute options for purchased components. A Purchased Components grid appears at the bottom of the page. Selecting this option categorizes purchased components for display purposes; no additional processing is performed.
Manufactured Components	Select to choose name and attribute options for manufactured components. A Manufactured Components grid appears at the bottom of the page. As with Purchased Components, selecting Manufactured Components categorizes

manufactured components for display purposes; no additional processing is performed.

- Routing Operations** Select to choose name and attribute options for routing operations. A Routing Operations grid appears at the bottom of the page. As with Purchased Components, selecting Routing Operations categorizes routing operations for display purposes; no additional processing is performed.
- Delta Information** Requests the display of the differences between this configuration and the last submitted configuration. Differences include additions, deletions, and changes to selection points, expressions, and externs.
- Compound Violations** Select to return a list of configuration violations for a solution based on a compound model. This check box is displayed only when you select a configuration type of Compound on the Display page.

Configuration Details

Options in this section of the page determine what information about the configurations that are generated under this schema will be returned to, and stored in, Order Capture. The first three fields—Components, Connections, and Structure—are available only when you select a configuration type of Compound on the Display page.

- Components** Select *All Components* or *None*.
- Connections** Select *All Connections*, *Filtered List*, or *None*.
- Structure** Select *Include Structure* or *None*.
- Selections** Select *All Selections*, *Filtered List*, or *None*.
- Expressions** Advanced Configurator uses Boolean, date, string, and numeric logic as key parts of its configuration capabilities. Select this check box to return values that are calculated by expressions (in the model) during the configuration session.
- External Variables** PeopleSoft Advanced Configurator can retrieve external data for the configuration session at runtime. Select this check box to return external value details during the order capture configuration session.
- Configuration Attributes** Configuration attributes data is normally not essential to the function of the Configurator, and includes data such as a person's name, phone, or email. Select this check box to return configuration attribute data at runtime.

Display Options

- Display Component Violations** Select to return component violation information at runtime.
- Display Selection Violations** Select to return selection violation information at runtime.

Package Components

The Package Components grid appears when you select the Package Component check box in the Request the Following section, which is described previously. If the product is a package item, you can select which of the package components and their attributes to include in the request details.

Accessing the Advanced Configurator Solution from Within PeopleSoft CRM

This section provides an overview of how to access Advanced Configurator, a sample product configuration, and discusses how to view configuration details:

Understanding How to Access Advanced Configurator

You can access the Advanced Configurator from:

- PeopleSoft Enterprise Order Capture.
- PeopleSoft Enterprise Order Capture Self Service.
- PeopleSoft Installed Products.
- Product Enterprise Catalog (Product Details).

The following table identifies the Collaborative Selling entry points from which you can access PeopleSoft Advanced Configurator:

Application/Access Point	Page Name	Navigation
Order Capture (RO_CAPTURE)	Entry Form Order (RO_FORM)	Create Order/Create Quote, Entry Form Add a (configurable) product to the order line, and click the Configurator button.
Order Capture Self Service (RE_CART)	Shopping Cart (RE_CART)	Add a (configurable) product to the shopping cart, and click the Configurator button.
Product Catalog (RB_CATALOG)	Product Details (RB_PROD_DTL)	Access a product catalog, select a product from the Product Display page, and click the Configurator button.

Users can configure their products and product packages by clicking the Configurator button within the calling application. When the product is configured, the configuration is saved to the database. This occurs for both simple and compound configurations. When the configuration session is complete, and order information is updated, the system returns the user to the main calling application.

Page Used to Access the Advanced Configurator Solution from Within PeopleSoft CRM

Page Name	Object Name	Navigation	Usage
Configuration HTML Page	CFG_HTML_SEC	<p>Create Order/Quote, Entry Form</p> <p>Add (configured) Product to Lines. Click the  button.</p> <p>Add (configured) Product to Cart, Shopping Cart. Click the  button.</p> <p>Service Management, Maintain Service, add (configurable) product, click the  button</p> <p>360 Degree View, Browse Catalog, Product Application page, Select configurable product, Click the  button.</p> <p>Access a product catalog, select a product from the Product Display page, click the  button.</p>	Configure a product using a PeopleSoft Advanced Configurator session.

Sample Product Configuration

The following example illustrates a configuration session that is initiated from within PeopleSoft Order Capture. It shows the custom user interface for a complete sample solution that is supplied with Advanced Configurator.

The screenshot shows a configuration user interface for a refrigerator. The interface is divided into several sections:

- *What will this freezer be used for?**: Radio buttons for Food, Ice, and Medical.
- *Refrigerator Type**: Radio buttons for Cooler and Freezer.
- *Refrigerator Dimensions**: Dropdown menus for Height, Width, and Depth, each with the text "Specify a height ...", "Specify a width ...", and "Specify a depth ..." respectively.
- *Cooling Unit**: Radio buttons for Cooler w/ Semi-Hermetic Compressor and Cooler w/ Hermetic Compressor [+ \$4,500.00].
- *Floor Surface**: Radio buttons for Concrete Floor, Epoxy Floor, Insulated Floor, and Tile Floor.
- *Wall Color**: Four color swatches (grey, white, purple, brown).
- *Thermostat**: A temperature gauge and a digital display showing 18.8. The display also shows "TAR" and "ELREHA" below the temperature.

The interface includes "Update", "Reset", "Cancel Changes", and "Return to Order" buttons at the top and bottom, and a "List Price: \$8,240.00" displayed on the right side.

Example of a configuration user interface

Viewing Configuration Details

Advanced Configurator enables you to extract information from individual configurations for additional processing and record-keeping. A common use of configuration details is the populating of the line details of an order or quote. When you set up the schema for the solution, you specify whether you want to extract configuration information and which data you want. Advanced Configurator delivers the data in XML form. Because you also specify an XSLT or stylesheet in the schema, the XML formatted data is rendered in a meaningful form.

Line Details
Find | View All
First ◀ 1 of 1 ▶ Last

Product Custom Walk In Freezer	Line 1	Total Price 23460.00	
Product ID 9999	Add Note	Total Recurring Price 00.00	
*Unit of Measure Each		List Price 25,500.00	
Order Qty 1.0000	Availability Check Failed	Discount Taken 2040.00	
Unit Price 0.00	View Adjustments	Discount Percentage 8.00	
Promotion Code		Minimum Price 22,100.00	
Shipment Single Shipment			

Cross/Up Sell Opportunities

			Product Description	Product ID	Relationship Description
			Refrigerator, Plastic Bins	10000	Alternates
			Air Cond, Fan	10010	Alternates

Configuration and Attributes

Product Brand Arctic King	Category Commercial Room
Model Number RV102	

Custom Walk In Freezer

Example of configuration details used to populate an order

PART 6

Building a Custom User Interface

Chapter 23
Understanding the Runtime System

Chapter 24
JSP and Page Templates

Chapter 25
Processing User Picks and Entries

Chapter 26
Processing Configurator Form Controls in JSP Pages

Chapter 27
Using JSP Form Control Templates

Chapter 28
Using the Page Editor Extensions for Dreamweaver

Chapter 29
Compound Modeling

CHAPTER 23

Understanding the Runtime System

This chapter discusses:

- Deployment framework.
- Advanced Configurator web components.
- Sequential application JSP pages.
- Deployment for a web application based on a single component model.
- Deployment for a solution in a mobile environment.
- Deployment for a web application based on a compound model.

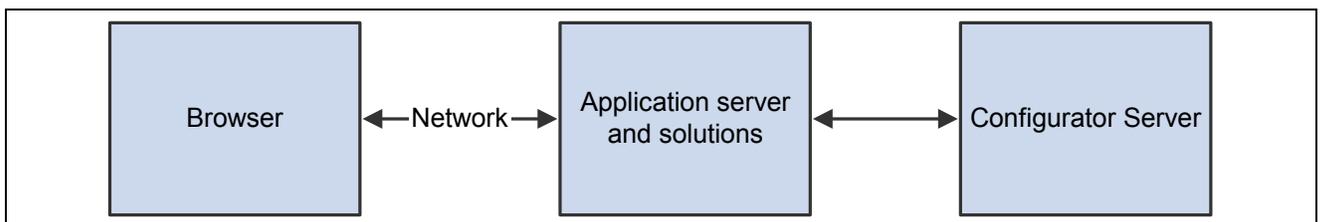
Deployment Framework

The PeopleSoft Advanced Configurator web deployment framework enables both rapid web-application development and good web performance.

The web application framework requires Java scripting and HTML coding skills; however, it separates the functionality into small components, or pages, that you can quickly compose and easily maintain.

The PeopleSoft Advanced Configurator engine is decoupled from the web application service, keeping the engine stateless and the entire state of each user's configuration session maintained solely by the web application. This is done by embedding all the information needed to recreate each user's session within hidden input fields on each page of the web application. This information is then sent back to the web server with every HTML form submission.

The Configurator engine is accessed through its public interfaces, which run on the application server. The following diagram shows the architecture of a midtier application managing the data flow between the Configurator engine service and the web user, in which the midtier proxies as the client:



User environment is decoupled from the processing

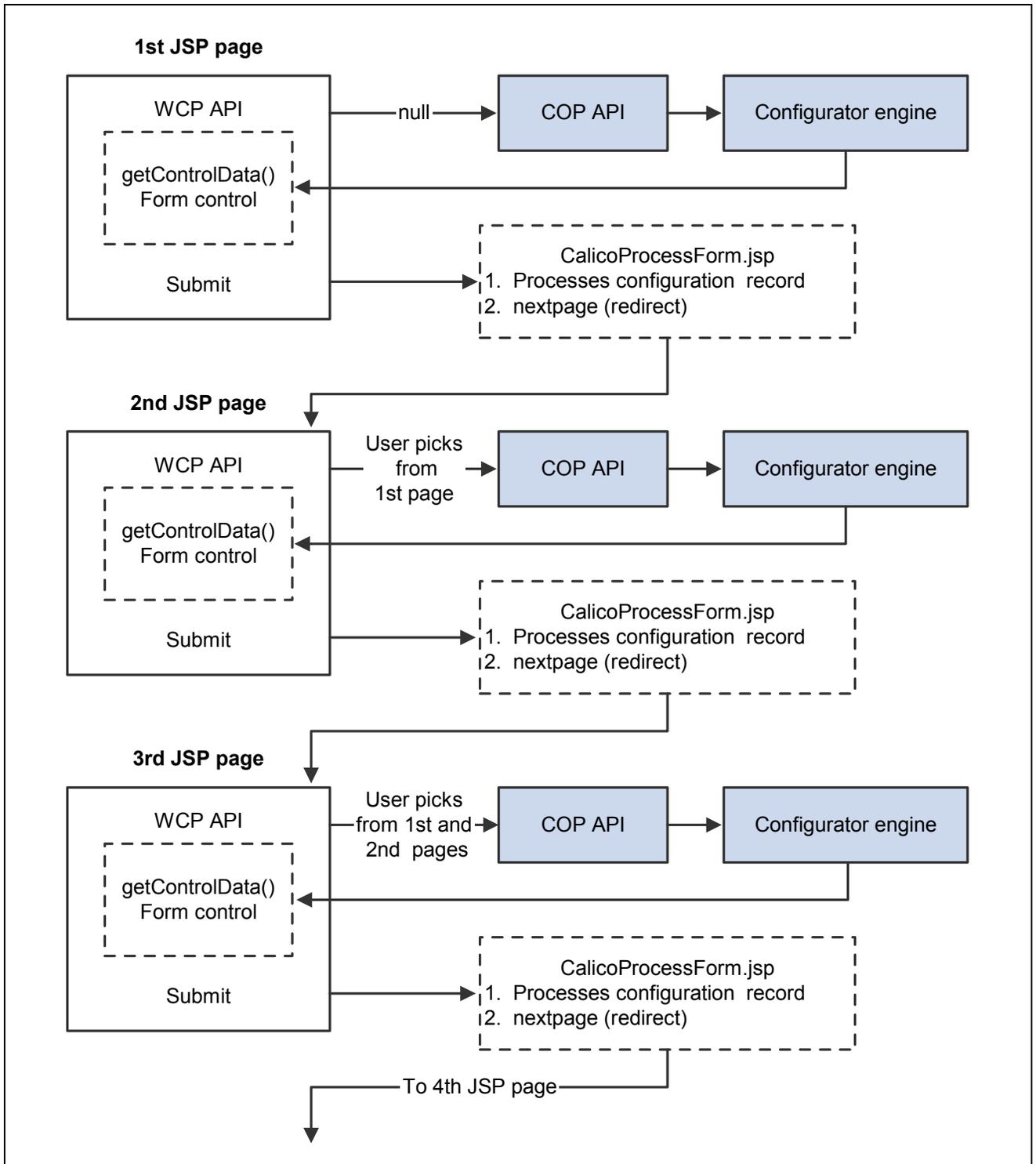
Create a Web client application for the Configurator by creating application JavaServer pages (JSP) that also run on the application server. Your JSP application pages call the Configurator interfaces and run other supporting Configurator JSP pages. JSP pages are web pages written using both Java scriptlets and HTML syntax.

See [Chapter 24, "JSP and Page Templates," page 307](#).

You can design your web pages entirely using HTML, except for the form controls that access data from the configuration models that are processed by the Configurator Engine. Each decision point in the model maps to a selection point for the client, or web application, and can be implemented by a PeopleSoft-specific HTML form control. Simple Java scriptlets in the JSP pages are necessary to manually include the Configurator form control templates, which implement the HTML selection points.

Note. For even faster and easier application development, you can use the popular Dreamweaver web-authoring tool's extensions to drag-and-drop Configurator's form control templates into your application pages.

During the configuration session, the user steps through a series of HTML form-based JSP pages, submitting a set of picks, or configuration records, for each page. The following diagram illustrates how the user's picks are sent to the Configurator engine, which processes the model's constraints and redirects the configuration state back to the subsequent JSP page:



Data submission and return

Form Controls

HTML provides these user-selection input/output mechanisms for related choices (or picks), such as:

- Option buttons, enabling the user to make a single selection from the displayed items; for example, the domain members of a decision point.

- Check boxes, enabling the user to make more than one selection from the displayed items
- Drop-down list boxes, enabling the user to make a single selection from a list of items.
- Selection lists, enabling the user to make more than one selection from a list of items.

Form controls are rendered by all the code between the <FORM> and </FORM> HTML tags.

Form Control Item

A user-selectable choice on a form control that maps to a domain member in a decision point in the model.

JSP Pages (JavaServer Pages)

Scripted pages that consist of both HTML and simplified Java that is scripted along with the HTML. JSP pages are processed into pure HTML with runtime data included. A JSP page is a file or URL with a .jsp extension instead of an .html extension.

See <http://java.sun.com/products/jsp/http://java.sun.com/products/jsp/>

Application JSP Pages

JSP pages that correspond to web pages that you create as a client application, which runs on a midtier environment, not in a browser. The JSP environment represents a client application connecting the web user to one or more server-side services.

Page template

An application JSP page that is processed to dynamically generate data to return to the web user along with the HTML web page. The data typically is provided by a specialized service; for example, the PeopleSoft Advanced Configurator Engine.

Form control template

A relatively small JSP page (provided) representing a form control that is included during the processing of an application JSP page.

Note. Configurator form control templates display selection points for a model on the users' web pages and accept users' picks.

Processor pages

JSP pages (provided) necessary to preprocess the user's picks before sending them to the Configurator Engine.

Advanced Configurator Web Components

The following interfaces enable you to create a dynamic web client application for the PeopleSoft Advanced Configurator Engine:

- Configurator form control templates: JSP pages that retrieve information from the Client Operations Processor (COP) about decision points and domain members, and dynamically generate HTML for user input and messages on pick violations.
- CalicoUI.properties: A text file that sets display properties for the HTML generated by Configurator form control templates.
- Configurator JSP processor pages: Enable you to use the Configurator controls templates from your application JSP pages.

- Web Client Processor (WCP): Converts user picks from strings to objects, and passes them to the COP for processing.
- Client Operations Processor (COP): Processes user picks to retrieve the current state of decision points and domain members from the engine.

See [Chapter 12, “Using the COP Java API,” page 161](#).

A dynamic Web client application is created by including Configurator JSP pages (among them, the Configurator control templates) in your application JSP pages, and setting template parameters and display properties for the form control template JSP pages.

The Configurator JSP pages make the necessary calls to the WCP and COP to generate an HTML page that has:

- Option buttons, check boxes, selection lists, and drop-down list boxes that display the current state of selection points and domain members.
- Text lists that display messages about constraint violations.
- A hidden INPUT tag that submits all previous user picks when the page is submitted.

Note. The Web client’s request for the first JSP page has no user picks, so the hidden INPUT tag has and submits none.

Sequential Application JSP Pages

In building an application, assume that:

- Page1.jsp contains the code for grouped radio buttons that let the Web client user choose either a coupe or a pickup automobile.
- Page2.jsp contains the code for grouped radio buttons that let the Web client user choose one of these interiors: leather, cloth, or vinyl.
- Neither page has default picks.
- The configuration model for the two pages, Vehicles01, has this constraint: a pickup cannot have a leather interior.
- When the user submits his/her picks on Page1.jsp, the next page, Page2.jsp is processed.

If the web client requests Page1.jsp, the special servlet for Page1.jsp creates an HTML page with radio buttons for *coupe* and *pickup* displayed as selectable—that is, neither selected nor eliminated. If the web client user chooses *pickup* and submits the HTML page, the servlet (the CLASS file of a JSP page) for Page2.jsp creates an HTML page with the radio buttons for cloth and vinyl displayed as selectable, and the radio button for leather displayed as computer-eliminated. If the web client user chooses a cloth interior and submits the second HTML page, a hidden INPUT tag submits the user’s previous pick of a pickup, while the radio button submits the user’s pick of a cloth interior.

Note. A JSP page that has both groups of radio buttons has the same effect when submitted: that is, if the user chooses a pickup, the hidden INPUT tag stores that choice and the radio button for leather is displayed as computer-eliminated.

Deployment for a Web Application Based on a Single Component Model

A component model is a single, standalone representation of a product or a component of a product that contains relationships based on its own objects rather than those of another model. A set of component models whose relationships are interdependent is called a compound model.

To deploy a solution:

1. Create a directory for your application JSP pages in the following directory:

```
\\bea\weblogic81\config\CalicoDomain\applications\  
CalicoApp\solutions
```

Note. It is good practice to create your application directory name only of lower-case characters.

The application JSP pages belong in an application folder; by default the application folders should be in the CalicoApp directory. The application server looks for application JSP pages according to the value resulting from a concatenation of the Name and Path attributes of the Application element in the config.xml file for the Configurator domain:

(\\bea\weblogic81\config\CalicoDomain\config.xml) in the format Path + \ + Name

2. Copy these two files in the same application folder:

CalicoProcessForm.jsp

CalicoUI.properties

3. Modify \\bea\weblogic81\config\CalicoDomain\config.xml to the desired settings.

See <http://e-docs.bea.com/wls/docs81/index.html>

and

PeopleSoft Enterprise CRM 9 Installation Guide

4. Modify \\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\Web-inf\config\verify.properties by setting the “messages” flag to FALSE.

5. You are responsible for creating or modifying only the following files for an entire Configurator web application:

- a. You must implement your application JSP pages.
- b. You should modify the display configuration file, CalicoUI.properties.
- c. You can customize the application’s copy of CalicoProcessForm.jsp.
- d. You can copy one of the form control templates (rename the copy, but put it in the application directory), and then customize the copy.

Optimizing Performance

For better performance, you should disable session support by starting your JSP application pages (that do not access the implicit session object) with the following tag:

```
<%@ page session="false" %>
```

This will prevent the JSP compiler from generating code to create session objects. However, you will not be able to implement any supporting JavaBeans with session scope for these applications.

Restore Policy

A user's request to recover a saved configuration presents a challenge in light of the requirement for that configuration to run on a compatible version of the model that it was created with. It is conceivable that the picks on the saved configuration correspond to selection points that were removed in an intervening model update.

Deployment for a Solution in a Mobile Environment

A solution based on PeopleSoft Advanced Configurator can be installed and run entirely locally. This capability ensures that a field user does not require a configuration-time connection to the enterprise installation of Configurator.

PeopleSoft Enterprise Mobile Configurator enables the field user to:

- Log on to the enterprise server to download a current Solution Package—a model(s), pages, and images that comprise the application.
- Configure products or services while offline.
- Log on to the enterprise and upload saved configurations and their associated artifacts (such as a bill of materials) to the enterprise database store or for further processing within an ERP system.
- Request existing configurations while logged into the enterprise system.
- Restore existing configurations for reference or modification.

To run a Solution Package, the mobile user's machine must meet these requirements:

- Windows 2000 operating system.
- The following Mobile Configurator components installed:
 - Configurator
 - WebLogic Application Server
 - Configurator Mobile Client

Configurator Mobile components are available on media separate from the Enterprise Server. Mobile machine installation is performed by an IT administrator.

Distributable Solution Packages are created and managed with the PeopleSoft Advanced Configurator Packaging Tool.

Very little is required to adapt a PeopleSoft Solution for mobile deployment.

The model implementor must see that all model data is internal to the model, and not requested from a database or another file. Such data can be easily internalized by the Visual Modeler.

The site developer must set the restore policy and return page in the CalicoUI.properties file. Restore policies are described in the following section.

See Also

[Part 2, “Product Modeling with a Component Model,” page 9](#)

[Part 3, “Product Modeling with Compound Models,” page 121](#)

[Part 7, “Mobile Product Configuration,” page 385](#)

Deployment for a Web Application Based on a Compound Model

To deploy a compound model-based solution:

1. Deploy the compound model’s component models (the .cms files) to the server by compiling them on the Visual Modeler or deploying them using the Administration Tool.
2. Similarly, deploy the compound model’s XML compound structure document to the server by compiling it on the Visual Modeler or deploying it using the Administration Tool.
3. Create a directory for the compound model in the following directory:


```
\\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\
```
4. Following the procedure described in “Create a directory for your application JSP pages in the following directory”, deploy, in the compound model directory you just created, each component model as if it is a single, standalone model.

When you are done, the directory structure should look similar to this example:

```
\\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\
<Compound Model>
\Model1
  \pages
    CalicoProcessForm.jsp
    CalicoUI.properties
\Model2\
  \pages
    CalicoProcessForm.jsp
    CalicoUI.properties
\Model3\
  \pages
    CalicoProcessForm.jsp
    CalicoUI.properties
```

5. Update the compound model properties file CalicoNA.properties with the appropriate values.

See [Appendix D, “Compound Model Properties File,” page 463](#).

6. Place CalicoNA.properties in the compound model’s root:

```
\\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\
<Compound Model>
CalicoNA.properties
<compound model schema>.xml
```

```
...  
\Model1  
\pages  
...
```


CHAPTER 24

JSP and Page Templates

This chapter discusses:

- The midtier framework.
- Scope of the servlet.
- Using JSP processing.
- Writing JSP.
- Using generated Java and class files.

The Midtier Framework

The source code for the midtier applications is put into a set of text files that resemble HTML pages. Each file, or page, contains the HTML code for the generic style of the web server's HTTP response to the types of HTTP requests from the user. These generic response pages are called page templates.

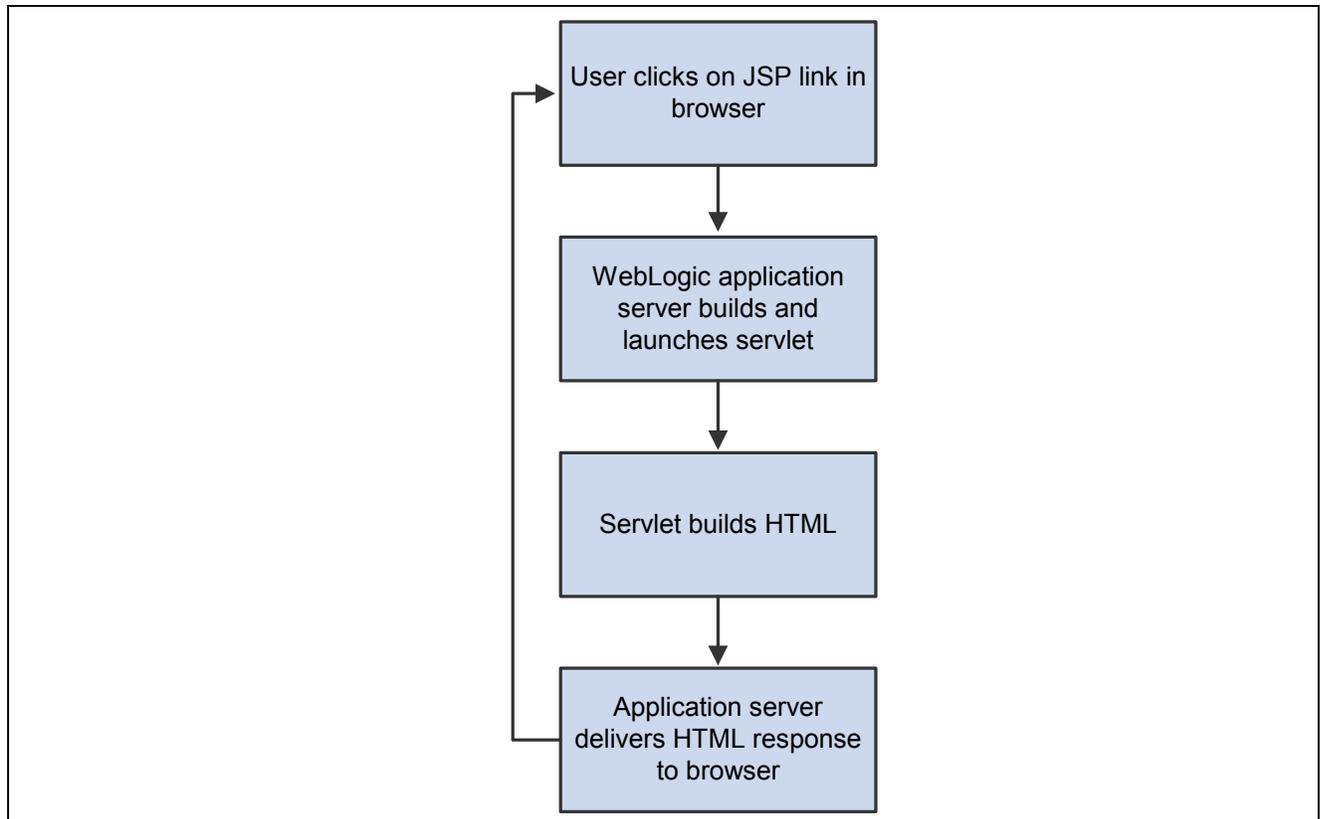
The emergence of JavaServer Pages (JSP) technology enables web page designers to intersperse Java source code with the HTML code to script server-side servlets that will compile and execute on the midtier host at run time. JSP includes the following languages:

- Java (scriptlets)
- HTML and JavaScript
- JSP tags

Note. JSP does not include, but does allow, client-side JavaScript, which is treated the same as HTML.

The web server passes incoming user requests to an application server specified by the port number portion of the URL. The application server converts the JSP source into pure Java source (for a servlet implementation). Then the application server runs the newly created servlet in its servlet engine. The output of the servlet is a stream of HTML text that the application passes to the web server to return to the user's browser.

The following diagram shows the flow of communication between the midtier and the user:



Flow of communication between the midtier and the user

PeopleSoft Advanced Configurator's framework employs the WebLogic application server in its environment. WebLogic application servers by default employ the port number 7777, but may be set to listen to the standard port 80.

Note. An external web server can be configured to proxy requests for JSP pages to the application server using port 7777.

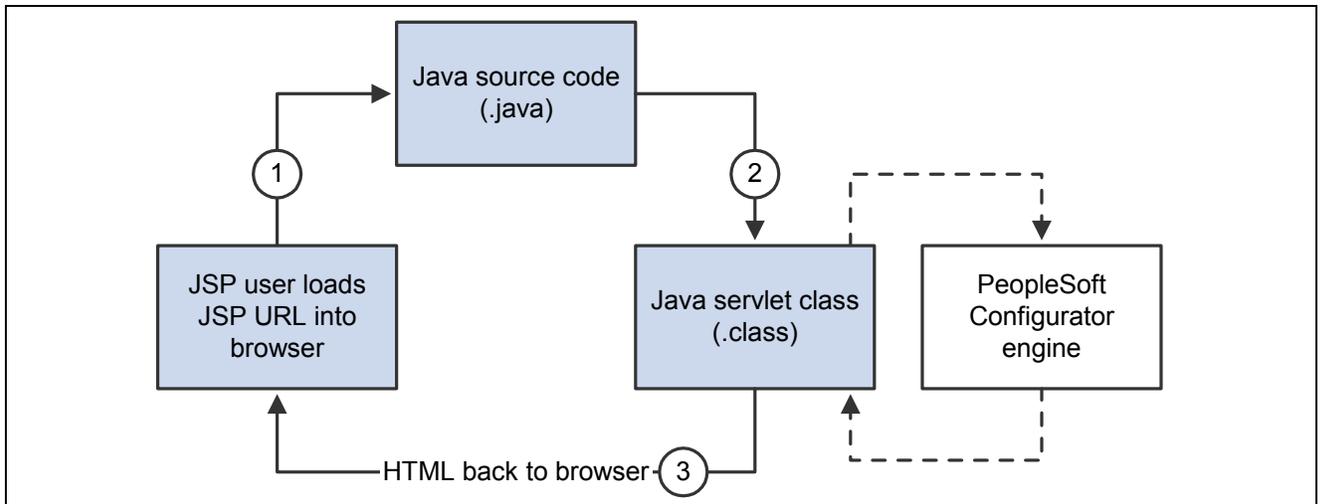
Users would load the JSP pages using the web server's host name (and port number if not port 80), as shown in the following URL:

```
http://WebServerHostName:7777/sample/myPC-page1.jsp
```

Whenever the user loads a JSP page in a browser, the WebLogic application server converts the JSP into a servlet JAVA file and a corresponding servlet CLASS file that implements the servlet interface. These files live securely on the machine hosting the WebLogic server, not in the browser. The WebLogic server then runs the servlet, which generates HTML that is served back to the browser. During a Configurator session, the user is linking to and loading JSP pages, yet only sees HTML output in the browser.

Note. You can compare the JSP source (on the midtier server) with the generated HTML (in the user's browser: use the View/Page Source menus).

The following diagram shows the transition from JSP source code to an executing servlet:



JSP executes on the midtier and displays HTML in the browser

Note. This compilation only occurs the first time that the JSP page is requested; however, if that JSP source file is modified, then WebLogic recompiles it.

Scope of the Servlet

Each page template defines a separate servlet that executes the complete Configurator response to an HTTP user request. Each servlet binary remains cached to rapidly respond to a particular request from any subsequent user or any subsequent session.

The Configurator's midtier servlets are like any other servlets—their scope is not limited to the source in a single JSP page or file. The Java code can call out to objects and methods in external CLASS or JAR files anywhere in the servlet CLASSPATH.

Best practices for JSP design use a scripting philosophy, keeping the page size small and complexity per page at a minimum. This enables rapid code development/deployment and easier maintenance. However, JSP-based applications need not be small or simple. JSP uses special directive tags to call out to JavaBeans or other components where much of the application logic can be delegated (and be transparent to the JSP source).

Using JSP Processing

A JSP source file is processed in two stages—translation time and request processing time. At translation time, which occurs when a user first loads a JSP page, the JSP source file is compiled to a Java class, usually a Java servlet. The HTML tags and as many JSP tags as possible are processed at this stage, before the user makes a request.

Request processing time occurs when the user clicks in the JSP page to make a request. The request is sent from the client to the server by way of the request object. The JSP engine then executes the compiled JSP file, or servlet, using the request values the user submitted.

When you use scripting elements in a JSP file, you should know when they are evaluated. Declarations are processed at translation time and are available to other declarations, expressions, and scriptlets in the compiled JSP file. Both expressions and scriptlets are also evaluated at translation time. The value of each expression is converted to a string and inserted in place in the compiled JSP file. Scriptlets, however, are evaluated at request processing time, using the values of any declarations and expressions that are made available to them.

The JSP page finally responds to the request as the source reads sequentially; that means that each block of Java code—and its resulting display—fires in the normal sequence of the HTML page, just as you’ve written it.

Writing JSP

JSP programming is beyond the scope of this document; however, you can access a very useful online tutorial for JSP at the following URL:

```
http://e-docs.bea.com/wls/docs81/jsp/index.html
```

You can comfortably read JSP code simply by knowing how the Java code can be embedded within the HTML, and how the HTML code can be embedded within the Java.

- The scriptlet syntax `<% . . . %>` can handle declarations, expressions, or any other type of code fragment valid in the page scripting language, e.g., Java.
- The include syntax `<%@ include file="anotherJspFile" %>` includes a specified external JSP page.
- JSP uses special tags to embed Java code within the HTML: The expression syntax `<%= . . . %>` defines a scripting language expression and casts the result as a string.
- The declaration syntax `<%! . . . %>` declares variables or methods.
- When you mingle scripting elements with HTML and JSP tags, you must always end a (JSP) scripting element before you start using (HTML) tags and then reopen the scripting element afterwards, as shown in the following JSP code:

```
<%
    if (condition) {
%> <!-- closing the scriptlet before the HTML tags start -->
    <!-- HTML text goes here outside the scripting tags...-->
<%    }
    else {
%> <!-- closing the scriptlet before the HTML tags start -->
    <!-- HTML text goes here outside the scripting tags...-->
<%    }          %> <!-- reopening a scriptlet to close the else {} block -->
```

- HTML text can be embedded within print statements in the Java code portions, for example

```
out.println("<P><BR><BR>collection:" +
currCollectionProperty.getName()+"</P><BR>");
```

You can have multiple blocks of Java code throughout a JSP page. You can switch between HTML and Java code anywhere, even within Java constructs and blocks. Notice in the loop shown here that declares a Java loop, switches to HTML, then back to Java to close the loop. The HTML is output multiple times as the loop iterates.

The following JSP code generates the typical “Hello World” HTML web page:

```

<html>
<head><title>Hello World in JSP</title></head>
<body>
<h1> Hello World Test </h1>
<p><i> This is HTML. The following is Java! </i><p>
  <%
    for (int i = 1; i<=5; i++)
      out.print("This is a Java loop! " + i + "Hello World<p>");
  %>
</body>
</html>

```

You can use the implicit object `out` to print directly to the servlet output stream from your Java code. Therefore, the servlet can output HTML text directly, as illustrated in the following line of JSP code:

```
out.print("This is a Java loop! <i>" + i + "</i> <p>");
```

Using Generated Java and Class Files

When the application server first compiles one of your JSP pages within your client application directory, the application server also creates the following folder and files, all named with a preceding underscore “_” character:

- A corresponding application directory by the same application name.
- A JAVA source file.
- A Java CLASS file.

WebLogic stores this new directory structure in:

```

\bea\weblogic81\config\CalicoDomain\myserver\.wlnotdelete\extract\
myserver_Calico⇒
Domain_CalicoApp\jsp_servlet

```

The application server also creates a corresponding JAVA source file and Java CLASS file with an underscore prefix for the Configurator JSP processor page.

For example, a Web client application contains these JSP pages:

```

C:\WirelessPlan\pages\myappdirectory\MyPage1.jsp
C:\WirelessPlan\pages\myappdirectory\CalicoProcessForm.jsp
C:\WirelessPlan\pages\myappdirectory\alicoUI.properties

```

The application server creates the following directory and files for saving the JAVA and CLASS files:

```

\bea\weblogic81\config\CalicoDomain\myserver\.wlnotdelete\extract\
myserver_⇒
CalicoDomain_CalicoApp\jsp_servlet\_myapp
directory\_myPage1.java
_myPage1.class
_CalicoProcessForm.java
_CalicoProcessForm.class

```

The application server generates only a Java source file and compiles it each time the Web client first requests a newly installed, or modified, JSP page.

CHAPTER 25

Processing User Picks and Entries

This chapter provides an overview of runtime processing and discusses how to:

- Initialize the web client processor (WCP).
- Process user picks and entries.
- Make Client Operations Processor (COP) calls.
- Use WCP methods.

Understanding Runtime Processing

The WCP communicates with the COP about the runtime configuration state of a PeopleSoft Advanced Configurator model. The COP is the public interface for the PeopleSoft Advanced Configurator Engine. It processes user picks to retrieve the current state of selection points and form control items.

The WCP interfaces are convenience application programming interfaces (API), and therefore are optional. You could implement the WCP behavior completely using the COP API.

The WCP API is used exclusively within the Configurator JavaServer Page (JSP)processor page, `CalicoStartFormInc.jsp`. The WCP has the following methods:

- `initialize`: Initializes the `WebClientOperations` object and starts a configuration session for the given model. This method has four signatures. Two of them take a `ClientOperations` object as an argument.
- `processConfigurationRecords`: Converts a configuration records string to `Choice` objects and passes those objects to the COP for processing.
- `getClientOperations`: Returns the `ClientOperations` object for the current configuration session. This enables calls to the COP.
- `getObjectNames`: Returns a string array of all of the decision points or selection points in the current model.
- `getConfigurationRecords`: Returns a configuration records string for the current configuration session.
- `loadConfigurationRecords`: Converts a stored configuration records string to `Choice` objects and passes those objects to the COP to load a configuration.
- `getModelName`, `getModelVersion`, and `getModelCompileVersion`: Return the requested information, as a string, for the current configuration session.
- `resetConfiguration`: Clears the internal state for the current configuration session.
- `release`: Releases the `WebClientOperations` object.

See Also

[Chapter 12, “Using the COP Java API,” page 161](#)

[Chapter 13, “Understanding the Configurator XML Interface,” page 185](#)

Initializing the WCP

To initialize the WCP object created by the application server, call the WCP method `initialize`, which has four signatures.

All four signatures take the following arguments:

- Name and version of the model.
- The locale of the form control data.
- Whether you want the control data to be HTML-encoded.

Two of the signatures also take as an argument the compile version of the model. Two signatures—one that takes a compile version argument and one that does not—also take a `ClientOperations` object as an argument.

Initializing the WCP with your own `ClientOperations` object enables you to extend the normal `ClientOperations` processing by implementing the `ClientOperations` interface with your own custom class and using an instance of that class to initialize the WCP. There are many reasons for extending the normal `ClientOperations` processing. One is to perform additional processing before or after normal processing. Another is to track and perform processing on nonconfiguration data that may be included on your JSP page, but that is not part of the actual model.

See [Chapter 12, “Using the COP Java API,” page 161](#).

Syntax

The following is the syntax to initialize the WCP:

```
void initialize (Locale appLocale,  
String modelName String modelVersion  
boolean HtmlEncoding)
```

```
void initialize (Locale appLocale,  
String modelName String modelVersion String modelCompileVersion  
boolean HtmlEncoding)
```

```
void initialize (ClientOperations clientOperations  
Locale appLocale,  
String modelName String modelVersion  
boolean HtmlEncoding)
```

```
void initialize (ClientOperations clientOperations  
Locale appLocale,  
String modelName String modelVersion String modelCompileVersion  
boolean HtmlEncoding)
```

Processing User Picks and Entries

If a web client user picks an item using a Configurator form control and submits the pick, the system posts the pick to a Configurator JSP page, `CalicoProcessFormInc.jsp`, which bundles user picks into a string. The string includes not only all of the picks that the user explicitly submitted, but also all prior user picks (which are stored in a hidden HTML input tag).

See Also

[Chapter 27, “Using JSP Form Control Templates,” page 325](#)

Configuration Records

Each user pick in the string created by `CalicoProcessFormInc.jsp` is called a configuration record. There are two types of configuration records, one for selection points and one for extern variables. A selection point configuration record has three parts, separated by a tilde (~):

- Selection point.
- Form control item: The form control item that is picked by the user.

Note. A form control item represents a domain member in a selection point in the model.

- Quantity: The number of copies of the form control item that is picked by the user.

The string of user picks (configuration records) is called a configuration records string. Each record in the string is separated by a pipe (|). Thus, the configuration records string has the following syntax:

```
<selectionPoint>~<controlItem>~<quantity>|<externVariable>~<value1>~
=>
<value2>|<selectionPoint>~<controlItem>~<quantity>
```

For example, if a web client user selects the form control item called Coupe at the selection point called Vehicle, and also picks the form control item called Leather at the selection point called Interior, the Configurator form control JSP processor page creates the following configuration record:

```
vehicleSelection~Coupe~1.0|interiorSelection~Leather~1.
```

Processing

To determine the runtime state of a Configurator model, the COP processes the user picks that are passed to it as Choice objects.

The WCP method `processConfigurationRecords`:

- Takes as an argument the configuration records string that is created by the Configurator JSP page that bundles user picks.
- Converts each record in the configuration records string to a Choice object.
- Passes all of those Choice objects to the COP to determine the runtime state of the current Configurator model.

Syntax

The syntax for `processConfigurationRecords` is:

```
void processConfigurationRecords (String configurationRecords)
```

Attribute Records

Each text input entry in the string created by `CalicoProcessFormInc.jsp` is called an attribute record.

An attribute record has two parts, separated by a tilde (~):

- Text input name
- Text input value

The string of text input entries (attribute records) is called an attribute records string. Each record in the string is separated by a pipe (|). The syntax of the attribute records string is:

```
<textInputName>~<textInputValue> [|<textInputName>~<textInputValue>]+
```

For example, if a web client user enters the value *Coupe* into the text input control `Vehicle`, and also enters the value *Leather* into the text input control `Interior`, the Configurator form control JSP processor page creates the following attribute record:

```
Vehicle~Coupe|Interior~Leather|
```

Processing

The values in text input controls are kept separate from the configuration record because they are *not* passed to the Configurator engine. Instead, the `CalicoStartFormInc.jsp` uses the `setAttribute` method to retrieve the processed attribute record from the request object and save it to the configuration object of the COP. Although this information is not used directly in determining the configuration, it is stored as part of the configuration record for the purpose of integration with other applications (and databases).

Syntax

The syntax for `setAttribute` is:

```
void setAttribute(String textInputCtrl, String textInputValue);
```

Making COP Calls

As the public interface for the PeopleSoft Advanced Configurator engine, the COP processes the user picks to retrieve the runtime state of selection points and form control items.

The WCP method `getClientOperations` functions differently in different situations:

- If you initialized WCP with your own `ClientOperations` object (that is, with an instance of the `ClientOperationsImpl` class), it returns that object for the current configuration session.
- If you did not initialize WCP with a `ClientOperations` object, it creates and returns a default instance of the `ClientOperationsImpl` class for the current configuration session.

This method does not take an argument.

Using the `ClientOperations` object enables you to directly retrieve the runtime state of selection points and form control items through the COP.

See [Chapter 12, “Using the COP Java API,” page 161](#).

Syntax

The syntax for `getClientOperations` is:

```
ClientOperations getClientOperations()
```

Using WCP Methods

The WCP has other methods to help you design a web client application. The following sections describe these methods and provide their syntax.

Getting Decision or Selection Points

The WCP method `getObjectNames` returns an array of string objects containing either:

- All selection points in the current Configurator model.
- Public selection points in the current Configurator model.

This method takes a Boolean argument: `True` returns all selection points; `False` returns only public selection points. It does not return private selection points—that is, selection points that are not displayed to and cannot be selected by the web client user.

Syntax

The syntax for `getObjectNames` is:

```
String[] getObjectNames(boolean allObjects)
```

Getting and Processing Stored Configuration Records

The WCP method `getConfigurationRecords` returns a configuration records string with all user picks for the current configuration session. It takes no argument.

Syntax

The syntax for `getConfigurationRecords` is:

```
String getConfigurationRecords()
```

If you save a configuration records string in an external database, you can pass it to the WCP method `loadConfigurationRecords`, which does the following:

- Converts each record in the configuration records string to a Choice object.
- Passes all of those Choice objects to the COP to load a configuration.

Syntax

The syntax for `loadConfigurationRecords` is:

```
void loadConfigurationRecords(String configurationRecords)
```

Getting Model Name, Version, and Compile Version

The WCP methods `getModelName`, `getModelVersion`, and `getModelCompileVersion`, each return a string with the requested information for the current configuration session. None of them take an argument.

Syntax

The syntax for these methods is:

```
String getModelName()  
String getModelVersion()  
String getModelCompileVersion()
```

Clearing Model State

The WCP method `resetConfiguration` clears the state for the current configuration session. It does not take an argument.

Syntax

The syntax for these methods is:

```
void resetConfiguration()
```

Releasing the WCP

The WCP method `release` releases:

- The `ClientOperations` object, which is either the default instance of the `ClientOperationsImpl` class that was created by `getClientOperations`, or the instance of the `COPEExtensionImpl` class that you passed to initialize.
- Internal objects for the current configuration session.

This method does not take an argument.

Syntax

The syntax for these methods is:

```
void release()
```

CHAPTER 26

Processing Configurator Form Controls in JSP Pages

This chapter provides an overview of the JSP pages that process a Configurator form control and discusses:

- Configurator JSP page flow.
- Processing Configurator form controls.
- Using Configurator JSP pages in a solution.

Understanding Configurator Form Control Processing

To create a client application for the Configurator, create JSP pages that use Configurator JSP pages. There are two types of Configurator JSP pages:

- Form control template: A JSP page that creates an HTML form control (option button, check box, or selection list) for a selection point for your model.
- (Form control) processor page: A JSP page that helps process a Configurator form control.

You can use a Configurator form control template in your Web client application without modification, setting only its parameters and perhaps its display properties. You can also modify a Configurator form control template and use the modified template in your application.

See [Chapter 27, “Using JSP Form Control Templates,” page 325](#).

You must correctly use all of the Configurator form controls framework in your Web client application. Normally, the JSP processor pages require no modification. Together, they receive, process, and return the information needed to implement the Configurator form controls that you include in your application. Although you can call the WCP or COP directly in the JSP pages that you create, the Configurator JSP pages normally make all the necessary calls to these two interfaces.

Configurator JSP Page Flow

JSP page statements are interpreted in the order that they are written.

The following steps through the execution path by way of the Configurator JSP pages that are included in each application JSP page:

1. Initialization of settings in the application JSP pages, (such as including delta pricing in the display) with the JSP built-in `jspInit()` function.

- Application JSP pages execute the HTML <FORM> start tag, which specifies the name, method, and action HTML tag attributes as follows:

```
<FORM name="form" method="POST" action="CalicoProcessForm.jsp">
```

- Application JSP pages executes the JSP include directive as follows:

```
<%@ include file="/calico/CalicoStartFormInc.jsp" %>
```

Note. CalicoStartFormInc.jsp includes both CalicoConstantsInc.jsp and CalicoControlInc.jsp.

- Application JSP pages execute the include directive for a Configurator JSP form control template (only once per type of Configurator form control template, regardless of how many of that type of template you use in that application JSP page).

Note. There may be more than one Configurator form control template per application page.

- Application JSP pages ends its HTML form block by including CalicoEndFormInc.jsp.
- Application JSP pages uses the HTML end form tag, </FORM> on the line following the JSP include directive that includes CalicoEndFormInc.jsp.
- After the user clicks the Submit or Reset button, CalicoProcessForm.jsp processes HTTP POST information to generate the configuration records for the user's picks.
These picks are then processed by the COP and the Configurator Engine.
- The Configurator Engine runs the constraints on the user's picks, and then returns the new configuration state to the redirected next page application JSP page in the application sequence.

Processing Configurator Form Controls

The Configurator uses the following JSP pages to process Configurator form controls:

- CalicoProcessForm.jsp: Includes CalicoProcessFormInc.jsp.
- CalicoProcessFormInc.jsp: Constructs a string of user picks made on a Configurator form control, and redirects the application server to the next page of the Web client application.
- CalicoConstantsInc.jsp: Declares string constants that are used by every Configurator form control and several Configurator JSP processor pages.
- CalicoStartFormInc.jsp: Creates several hidden HTML input tags, sets display properties and pricing attributes for Configurator form controls, initializes WCP, and processes user picks.
- CalicoControlInc.jsp: Declares several methods that are used by every Configurator form control.
- CalicoEndFormInc.jsp: Creates hidden input tags on Configurator form controls and the next page, and releases the WCP object.

The next six sections describe these JSP pages in more detail.

Pre-Process Form Page

CalicoProcessForm.jsp does the following to help process Configurator form controls:

- Lets you preprocess information posted by the Web client before you pass it on to be processed by the other Configurator JSP pages.
- Includes CalicoProcessFormInc.jsp.

Post any form having a Configurator form control to CalicoProcessForm.jsp. If you do not alter this JSP page to preprocess information posted by the browser, it always includes CalicoProcessFormInc.jsp, which redirects to that page.

Process Form Page

CalicoProcessFormInc.jsp does the following to help process Configurator form controls:

- Constructs a string of user picks made on Configurator form controls and updates the configuration record.
- Constructs a string of attributes entered into text-entry controls and updates the attribute record.
- Redirects the application to the next JSP page of your Web client application.

Unless you preprocess information posted by the Web client, you effectively post to this JSP page.

CalicoProcessFormInc.jsp creates a configuration records string that the WCP parses to pass user picks to the COP for processing. This string includes two types of user picks:

- Previous user picks posted from a hidden HTML input tag.
- Current user picks explicitly submitted by the user (this action updates the configuration record).

Note. CalicoStartFormInc.jsp creates the hidden HTML input tag by aggregating explicit user picks.

CalicoProcessFormInc.jsp redirects to the page whose path you set in the name attribute of the HTML input tag that submits your application page with a Configurator form control. You set the path by using the “Redirect_” prefix; for example:

```
name="Redirect_/myjspapplication/Page1.jsp" type="submit" value="Next Page">
```

Reset a page in the same way, using the “Reset_” prefix, which clears all user pick. For example:

```
<INPUT name="Reset_/myjspapplication/Page1.jsp" type="submit" value="Clear">
```

Constants Page

CalicoConstantsInc.jsp declares string constants. This page is included in the following form controls processor JSP pages:

- CalicoProcessFormInc.jsp
- CalicoStartFormInc.jsp

Start Form Page

CalicoStartFormInc.jsp does the following to help process Configurator form controls:

- Creates hidden HTML input tags for the following:
 - Configuration records string
 - Attribute records string
 - UI version

- Current page
- Model information
- Initializes a WCP object, and processes the configuration records string.
- Gets a ClientOperations object.
- Reads the CalicoUI.properties file and sets the display properties for Configurator form controls.
- Sets pricing attributes for Configurator form control items.

Include this JSP page in every form that has a Configurator form control immediately after the <FORM> tag.

CalicoProcessFormInc.jsp creates the configuration records string by aggregating previous and current user picks. By creating a hidden HTML input tag for the configuration records string, CalicoStartFormInc.jsp ensures that previous user picks are submitted with explicit user picks, and effectively preserves the state of your Configurator model.

See Also

[Chapter 26, "Processing Configurator Form Controls in JSP Pages," Process Form Page, page 321](#)

Control Page

CalicoControlInc.jsp declares several methods that are used by every Configurator form control. These methods do the following for each Configurator form control:

- Add the form control to a form control map: register a Configurator form control.
- Get the model's control data, including elimination levels, for the form control.
- Determine if the form control is priced, and format delta pricing.
- Get the form control's display properties for conflicted and selectable.
- Parse the input string into an array of strings for each attribute of the domain member's display attributes.

End Form Page

CalicoEndFormInc.jsp does the following to help process Configurator form controls:

- Creates hidden input tags to later post the following information:
 - A map of Configurator form controls created as each form control appends its control ID using the mapping method declared in CalicoControlInc.jsp.
 - The next page.
- Releases the WCP object created in CalicoStartFormInc.jsp.

Include this JSP page in every form that has a Configurator form control, immediately before the </FORM> tag.

Using Configurator JSP Pages in a Solution

When you design a Web client application for the Configurator, you typically use the JSP pages provided, including the form control templates, and submit to or include them on your JSP pages without change.

To implement the Configurator JSP pages, you must do the following:

- Initialize your JSP page to include pricing information if you want it. Use the standard JSP built-in method, `JspInit()`.
- Include the Configurator form control templates that you want, where you want them on the page.
- Set the parameters for each form control template, including at least the following:
 - Object name, which is the model's decision point class name.
 - Form control ID.
 - Form control caption.
 - Domain member attributes (such as description, size, and price), which are displayable alongside the form control items.
 - Whether to provide the option of choosing none of the form control items, and if so, the text to use for displaying that choice: this is option for single-selection selection points.
- Modify the UI version, model name, and model compile version in `CalicoUI.properties`
- Set the HTML display properties (such as images, text, and text colors) for indicating the state of an item (such as user-selected) in `CalicoUI.properties`.

Note. This is optional. Change these configurations properties only if you would like to customize the HTML look-and-feel of your form controls (application scope).

- Include other `CalicoStartFormInc.jsp`, `CalicoEndFormInc.jsp`, and include Configurator form control templates where needed.

The following code example is an application JSP page that uses the Configurator interfaces:

```
<HTML>
<HEAD>
  <TITLE>Sample Page</TITLE>
</HEAD>
<BODY>
<%!
public void jspInit()
{
  // turn on pricing for controls
  setPricing(true);

  // set pricingControls to specify specific controls
  // that need to be priced and their pricing attributes.
}
%>
<FORM name="SampleForm" method="POST" action="CalicoProcessForm.jsp">
<%@ include file="/calico/CalicoStartFormInc.jsp" %>
<%@ include file="/calico/templates/html/SingleSelectGroup.jsp" %>
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null) {
    params.clear();
    params.put(PARAM_OBJECTNAME, "VehicleSelection");
    params.put(PARAM_ATTRIBUTES, "Description");
```

```
        params.put(PARAM_CAPTION, "Select a Vehicle");
        params.put(PARAM_CONTROLID, "VehicleSelectionCtrl");
        generateSingleSelectGroup(params, request, out);
    }
    %>
    <!-- METADATA TYPE="CalicoControl" ends span -->
    <%@ include file="/calico/CalicoEndFormInc.jsp" %>
</FORM>
</BODY>
</HTML>
```

The methods for setting pricing information mentioned in the code comments to `jspInit()` are declared in `CalicoStartFormInc.jsp`. They have the following syntax:

```
setPricing(boolean obtainPricingInformation)
setPricingControls(Map[] selectionPoints)
```

You can also set the range of the levels of eliminated items that you want to display. The commented code in the sample JSP page displays eliminated items having levels from 4 to 9, inclusive.

More specifically, you must do the following on each JSP page:

- Specify `action="CalicoProcessForm.jsp"` in the `<FORM>` tag.
- Include `CalicoStartFormInc.jsp` just after the `<FORM>` tag.
- Insert Configurator-related scriptlets and include all the Configurator form control templates after `CalicoStartFormInc.jsp`, but before `CalicoEndFormInc.jsp`.
- Include `CalicoEndFormInc.jsp` just before the `</FORM>` tag.

CHAPTER 27

Using JSP Form Control Templates

This chapter provides overviews of form control templates; properties, parameters, and attributes; and form control templates, and discusses how to:

- Use configuration form control templates.
- Specify the model and locale properties for the solution.
- Application page example.
- Configure a form control template.
- Register custom form control templates.
- Custom form control template example.
- Common errors.

Understanding Form Control Templates

Each decision point in the model maps to a selection point for the client and can be implemented by an HTML form control. This chapter describes the configurator templates for the form controls that are supported in the Configuration Client, which is the midtier application. A form control template is a separate Java Server Page (JSP) page that generates a form control, using the selection point data, for a web page. Configurator provides a JSP template file for each type of form control.

Your application JSP pages can contain form controls if you include their corresponding form control templates.

Note. All form control templates, Configurator-specific related scriptlets, and Java code should be inserted between the inclusion of the `CalicoStartFormInc.jsp` and `CalicoEndFormInc.jsp` processor pages. The JSP initializing method, `jspInit()`, must precede `CalicoStartFormInc.jsp`.

You can modify the appearance and behavior of the form control display in the following ways:

- Application scope: Modifying the definitions of display properties in a configuration file that an application page passes to the form control templates.
- Form control scope:
 - Setting parameters in the application pages.
 - Configuring a Configurator template: Extending the form control template by copying the JSP template, modifying that copy, and saving it as a configured template file that you can include in the application pages.

With the existing Configurator form-control templates, you can set either the HTML-level display properties, such as the text color of the display data and selection state of the form control item, or configure the presentation of the data of the form control.

You can specify the data display in the form control to:

- Display domain member attribute information.
- Sort the form control items.
- Eliminate form control items of eliminated domain members.
- Eliminate the form control items of the eliminated domain members whose elimination levels are beyond an inclusionary range.
- Display delta pricing and the total price.

Understanding Properties, Parameters, and Attributes

The presentation of a form control is determined by the display properties of the HTML for the application, by the included display properties and data of the HTML for the form control and its data, and by the display attributes of the domain member of the decision point.

Properties

The properties for a form control are the configurable display characteristics of the form control items that present their selection states and their delta pricing. All form controls properties are located in the CalicoUI.properties file. These properties are available application-wide, but to be used by a form control, they must be passed to the form control as parameters.

Parameters

The parameters for a form control are both HTML and data presentation characteristics that your application pages pass to included form control templates. Typically, the properties are passed to the templates as parameter values.

Attributes

The attributes for a form control are determined by the attributes in the model for the domain members. Typically, the attributes are passed to the templates as parameter values. Use the PARAM_ATTRIBUTES parameter to display the data for the specified attributes.

Understanding JSP Code Templates

PeopleSoft Advanced Configurator provides a JSP code template for all HTML-compatible form controls. This section describes the usage and behavior of each. JSP code and the HTML output is located in the referenced appendix for each.

Note. The filenames for the form control templates are in mixed case, that is, `SingleSelectGroup.jsp`. For the deployments running on Solaris systems, make sure the characters of the filename match correctly (case-sensitive) with the template filenames.

Single-Select Group

The single-select group form control *SingleSelectGroup.jsp* consists of a list of form control items, of which only one can be chosen. If one item in a list is selected, any previously selected item in the same list is deselected. The items of a form control are displayed as radio buttons.

You can create the following single-select group form control for the decision point `OSSelection`, with the caption *Operating System*, and the Generate None option.

Operating System

None

Windows 2000

Windows NT

A single-select group control

The following code generates the previous form control:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "[MyComputerConfig].OSSelection");
    params.put(PARAM_ATTRIBUTES, "Desc");
    params.put(PARAM_CONTROLID, "OSSelectionCtrl");
    params.put(PARAM_CAPTION, "Operating System");
    params.put(PARAM_GENERATENONEMODE, "1");
    params.put(PARAM_GENERATENONETEXT, "None");
    generateSingleSelectGroup(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

See [Appendix C, “Advanced Configurator Form Controls,” Single-Select Group Form Control, page 453.](#)

MultiSelect Group

A multiselect group form control consists of a list of form control items, of which more than one item can be chosen. The items of a form control are displayed as check boxes. Each item can be either selected or deselected.

You can create the following multiselect option group form control for the decision point `CDSelection` with the caption *Multimedia*.

Multimedia

CD 20X

CD 40X

DVDC

Multiple-select group form control

The following code generates the control in the figure:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "[MyComputerConfig].CDSelection");
    params.put(PARAM_ATTRIBUTES, "Desc");
    params.put(PARAM_CONTROLID, "CDSelectionCtrl");
    params.put(PARAM_CAPTION, "Multimedia");
    generateMultiSelectGroup(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

See [Appendix C, “Advanced Configurator Form Controls,” Multi-Select Group Form Control, page 454.](#)

Single-Select List

A single-select list form control consists of form control items in a drop-down list, of which only one can be chosen. It’s similar in functionality to a single-select group form control, but it requires less form control space for the display.

The StateTag indicates the domain member state of the form control item in the drop-down list. The following table shows the default text symbols that are used to connote the domain member states:

Symbolic tags	Domain member states
–	Selectable
>>	User-selected
>	Computer-selected
>	Default-selected
X	User-eliminated
X	Computer-eliminated
!	Conflict

Note. You can configure these state tags by modifying the property values in the CalicoUI.properties file, which is located in your application directory.

You can create the following single-select list form control for the decision point OSSelection with the caption *Operating System* and the None option.

Single-select list

The following code generates the form control in the figure:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "[MyComputerConfig].OSSelection");
    params.put(PARAM_ATTRIBUTES, "Desc");
    params.put(PARAM_CONTROLID, "OSSelectionCtrl");
    params.put(PARAM_CAPTION, "Operating System");
    params.put(PARAM_GENERATENONEMODE, "1");
    params.put(PARAM_GENERATENONETEXT, "None");
    generateSingleSelectList(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

See [Appendix C, “Advanced Configurator Form Controls,” Single-Select List Form Control, page 456.](#)

Multi-Select List

A multi-select list form control consists of form control items in a list, of which more than one item can be chosen. To select or deselect more than one item, a user presses the Ctrl/Shift keys and clicks the subsequent items to select them or clicks (selected items) to deselect them.

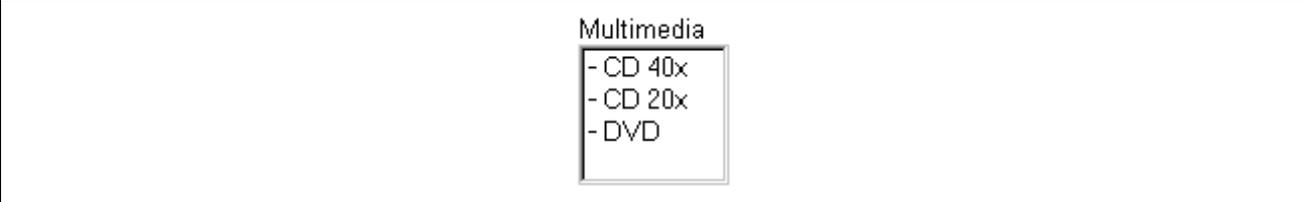
The StateTag indicates the domain member state of the form control item in the drop-down list. The following table shows the default text symbols that are used to connote the domain member states:

Symbolic tags	Domain member states
-	Selectable
>>	User-selected
>	Computer-selected
>	Default-selected

Symbolic tags	Domain member states
X	User-eliminated
X	Computer-eliminated
!	Conflict

Note. You can configure these state tags by modifying the property values in the CalicoUI.properties file, which is located in your application directory.

You can create the following multiselect list form control for the decision point CDSelection with the caption *Multimedia*.



Multiselect list control form

The following code generates the previous form control:

```

<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "[MyComputerConfig].CDSelection");
    params.put(PARAM_ATTRIBUTES, "Desc");
    params.put(PARAM_CONTROLID, "CDSelectionCtrl");
    params.put(PARAM_CAPTION, "Multimedia");
    generateMultiSelectList(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->

```

See [Appendix C, “Advanced Configurator Form Controls,” Multi-Select List Form Control, page 456](#).

Single-Select Table

A single-select table form control consists of form control items in the row or column format based on radio buttons and table HTML elements. Each row in the table contains a radio button and attributes for each available form control item. Only one of the form control items can be chosen at a time.

You can create the following single-select table form control for the decision point OSSelection with the caption *Multimedia* and the None option.

	Description	Part Number	Unit Cost
<input type="radio"/>	None		
<input type="radio"/>	Windows 98	MSFT-854	80.00
<input type="radio"/>	NT	MSFT-898	99.00

Single-select table form control

The following code generates the form control in the figure:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "[MyComputerConfig].OSSelection");
    params.put(PARAM_ATTRIBUTES, "Desc, PN, UC");
    params.put(PARAM_CONTROLID, "OSSelectionCtrl");
    params.put(PARAM_CAPTIONIMAGE, "/MyComputer/images/cd.gif");
    params.put(PARAM_COLUMNHEADINGS, "Description, Part Number, Unit Cost");
    params.put(PARAM_GENERATENONEMODE, "1");
    params.put(PARAM_GENERATENONETEXT, "None");
    generateSingleSelectTable(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

In the preceding code block, the PARAM_OBJECTNAME example value *[MyComputerConfig].OSSelection* is the name of a decision point in the model. *MyComputer* is the model name, appended with the string, *Config*. *MyComputerConfig* must be bracketed between [and] characters. *[MyComputerConfig] OSSelection* is the decision point name.

See [Appendix C, “Advanced Configurator Form Controls,” Single-Select List Form Control, page 456](#).

Multi-Select Table

A multi-select table form control consists of form control items in the row or column format based on check boxes and table HTML elements. Each row in the table contains a check box and attributes for each available form control item. More than one of the form control items can be chosen at a time.

You can create the following multi-select table form control for the decision point CDSelection with the caption *Multimedia*.

MultiMedia			
	Description	Part Number	Unit Cost
<input type="checkbox"/>	CD 40x	SONY-40	60.00
<input type="checkbox"/>	CD 20x	SONY-20	40.00
<input type="checkbox"/>	DVD	SONY-DVD	80.00

Multi-select table form control

The following code generates the form control in the figure:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "[MyComputerConfig].CDSelection");
    params.put(PARAM_ATTRIBUTES, "Desc, PN, UC");
    params.put(PARAM_CONTROLID, "CDSelectionCtrl");
    params.put(PARAM_CAPTION, "MultiMedia");
    params.put(PARAM_COLUMNHEADINGS, "Description, Part Number, Unit Cost");
    generateMultiSelectTable(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

See [Appendix C, “Advanced Configurator Form Controls,” Multi-Select Table Form Control, page 457.](#)

Single-Select Image

The single-select image template places an image on the page where you indicate, to select a domain member. Specify an image and location for each domain member of the selection point. At runtime, selecting one image deselects another. Include the template for the single-select image in the following way:

```
<%@ include file="/calico/templates/html/SingleSelectImage.jsp" %>
```

For example, you can create the following single-select image controls for a selection point with three domain members.



Two single-select image controls, before and after selecting an image

When you select and submit the selectable blue image, it is replaced by a corresponding selected blue image.

Note. Single-select images can be placed on the page where you specify. If you do not specify a location, they are placed on a line, bottom-aligned.

The following code generates the first (the blue image) of the two controls in the figure.

```

<!-- METADATA TYPE="CalicoControl" startspan -->
<%
if (params != null)
{
  params.clear();
  params.put(PARAM_OBJECTNAME, "[AudioConfig].kitSelection");
  params.put(PARAM_DOMAINMEMBERNAME, "DashSpacer");
  params.put(PARAM_IMAGENAME, "blue.gif");
  params.put(PARAM_IMAGEPATH, "SSImages");
  params.put(PARAM_AUTOSUBMIT, "true");
  generateSingleSelectImage(params, request, out);
}
%>
<!-- METADATA TYPE="CalicoControl" ends span -->

```

Parameters are:

- **PARAM_OBJECTNAME**: The unique object name (“<modelNameConfig>.decisionpointname”), which is used to retrieve the items of a form control.
- **PARAM_DOMAINMEMBERNAME**: The name of the domain member that the image selects.
- **PARAM_IMAGEATTRIBUTE**: The name of the domain member attribute whose value is the image that selects the domain member.
- **PARAM_IMAGENAME**: The name of the image that selects the domain member.

See [Appendix C, “Advanced Configurator Form Controls,” Single-Select Image, page 458](#) and [Chapter 27, “Using JSP Form Control Templates,” Parameters in the Inclusion Set, page 344](#).

Note. Provide either the image attribute or the image name, but not both.

Optional image parameters are:

- **PARAM_IMAGEPATH**: The Uniform Resource Locator (URL) path to the directory that has the images.
- **PARAM_IMAGEWIDTH**: The width of the image.
- **PARAM_IMAGEHEIGHT**: The height of the image.
- **PARAM_IMAGESTATES**: A comma-delimited list of states, other than selectable, for which alternative images exist whose names follow a naming convention.

The naming convention appends an underscore followed by one or two letters to either the image name or the value of the image attribute. For example, image1.gif becomes image1_s.gif and image1_c.gif. The comma-delimited list of states for which images exist uses the same letters (without the underscore) that are appended to the image name; for example, “us, es, e, c.”

Other possible states and the naming convention for their images are:

- Selected (**_s**): Use if you do not display computer-user variations.
- User-selected (**_us**).
- Computer-selected (**_cs**): Default-selected can share this image.
- Default-selected (**_ds**): Computer-selected can share this image.
- Eliminated (**_e**): Use if you do not display computer-user variations.
- User-eliminated (**_ue**).

- Computer-eliminated (*_ce*).
 - Conflicted (*_c*).
 - **PARAM_MOUSEOVERIMAGES**: “True” means that mouseover images are available. The default value is “False.”
- You must also provide mouseover images whose names follow a naming convention. The convention appends an underscore and the letters “mo” to the name of any image that changes when the mouse moves over it, for example, `image1_mo.gif` or `image1_us_mo.gif`.
- **PARAM_ALTTEXTATTRIBUTE**: The name of the domain member attribute whose value is the text that is used as an alternative for the image. Most browsers display this text as a tool tip when the mouse is over the image.
 - **PARAM_ALTTEX**: The text that is used as an alternative for the image, if the domain member does not have an alternative text attribute.
 - **PARAM_ADDITIONALATTRIBUTES**: Comma-delimited list of domain member attributes that are available to client-side script. Do not include the image attribute or the alternative text attribute.
 - **PARAM_AUTOSUBMIT**: *True* means that the form should be submitted when the user selects the image. The default value is *False*.

A JavaScript object, **calicoSSI**, is created for each image that is generated by the image templates, and is passed to JavaScript callback functions to identify the image that is being created, selected, deselected, moused over, moused off of, or auto-submitted. Each of these objects has these properties:

Property Name	Property Type	Description
<code>objectName</code>	String	The decision point of the control.
<code>ctrItemName</code>	String	The domain member of the item’s.
<code>state</code>	Number	1 if the image is selected, 0 if it is not selected.
<code>tagName</code>	String	Value of the <code></code> attribute for the item.
<code>attributes</code>	Array of Strings	An array having all of the attributes of the item.
<code>altText</code>	String	Alternative text for the image of the item.
<code>priced</code>	Boolean	True if the control is priced.
<code>delta price</code>	Number	The delta price of the item.

Optional callback parameters are:

- **PARAM_CREATECALLBACK**: The name of the JavaScript function that is called when a client-side single-select image object is created.

Parameter: the newly created object.

Return value: *None*

For example, `createCB()`:

```
function createCB(calicoSSI)
{
```

```

calicoSSI.myCustomImg = new Image(143, 126)
calicoSSI.myCustomImg.src = calicoSSI.attributes[0]
}

```

createCB() caches an image specified by a domain member attribute.

- **PARAM_SELECTCALLBACK**: The name of the JavaScript function that is called when a domain member image is selected.

Parameter: The object for the image that is being selected

Return value: *True* if the selection can proceed; otherwise, *False*.

For example, selectCB():

```

function selectCB(calicoSSI)
{
    document.images["myCustomImg"].src = calico.SSI.customImg.src
    return true
}

```

selectCB() updates an image with an image cached by createCB().

- **PARAM_UNSELECTCALLBACK**: The name of the JavaScript function that is called when a domain member image is deselected—that is, when another domain member image is selected.

Parameter: The object for the image that is being deselected.

Return value: “True” if the deselection can proceed; otherwise, “False.”

For example, unselectCB():

```

function unselectCB(calicoSSI)
{
    var retval = true
    if (someRareCondition(calicoSSI))
    {
        var retval = false
        alert("Explanation of problem")
    }
    return retval
}

```

unselectCB() may prevent deselecting the item, which would prevent selecting another item.

- **PARAM_MOUSEOVERCALLBACK**: The name of the JavaScript function that is called when the mouse moves over a domain member image.

Parameter: The object for the image that the mouse is over.

Return value: “False” if the function updates the browser’s status bar. The default value of “True” lets the alternative text for the image appear in the status bar.

For example, mouseOverCB():

```

function mouseOverCB(calicoSSI)
{
    window.status = calicoSSI.attributes[1]
    return false
}

```

mouseOverCB() puts text from a domain member attribute in the status bar.

- **PARAM_MOUSEOUTCALLBACK**: The name of the JavaScript function that is called when the mouse moves off a domain member image.

Parameter: The object for the image that the mouse is moving off

Return value: “False” if the function updates the status bar of the browser. The default value of “True” allows text to be cleared from the status bar.

For example, mouseOutCB():

```
function mouseOutCB(calicoSSI)
{
  window.status = "Eat at Joe's"
  return false
}
```

mouseOutCB() puts a static string in the status bar.

- **PARAM_AUTOSUBMITCALLBACK**: The name of the JavaScript function that is called before the form is submitted because an image is selected whose auto-submit attribute is “True.”

Parameter: The object for the image that is being auto-submitted Return value: “True” if the form can be submitted; otherwise, “False.”

For example, autoSubmitCB():

```
function autoSubmitCB(calicoSSI)
{
  return confirm("Submit your choices?")
}
```

autoSubmitCB() asks the user to confirm that the form can be submitted.

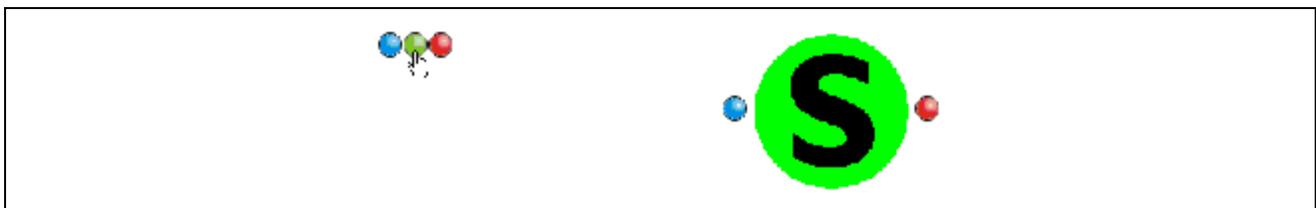
Single-Select Image Table

The single-select image table template places images for every domain member of a selection point either horizontally in a row, or vertically in a column on the page, to select among them. Specify an image for each domain member of the selection point, and whether you want them placed in a row or a column. At runtime, selecting one image deselects another.

Include the template for the single-select image table in the following way:

```
<%@ include file="/calico/templates/html/SingleSelectImageTable.jsp" %>
```

For example, you can create the following single-select image table control for a selection point with three domain members.



Single-select Image table, before and after you select an image

When the selectable green image is selected and submitted, it is replaced by a corresponding selected green image.

Note. The images of a single-select table image control are placed in a table row (in a table cell) and centered.

The following code generates the control in the figure:

```
<!-- METADATA TYPE="CalicoControl" startspan -->
<%
if (params != null)
{
  params.clear();
  params.put(PARAM_OBJECTNAME, "[AudioConfig].kitSelection");
  params.put(PARAM_IMAGEATTRIBUTE, "SSImage");
  params.put(PARAM_IMAGEPATH, "SSImages");
  params.put(PARAM_AUTOSUBMIT, "true");
  generateSingleSelectImageTable(params, request, out);
}
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

See [Chapter 27, “Using JSP Form Control Templates,” Parameters in the Inclusion Set, page 344.](#)

The optional table parameters are:

- **PARAM_IMAGEATTRIBUTE:** The name of the domain member attribute whose value is the image that selects the domain member.
- **PARAM_COLUMNS:** The number of columns in the table. The number of domain members that are displayed at runtime determines the number of rows.
- **PARAM_ROWS:** The number of rows in the table. The number of domain members that are displayed at runtime determines the number of columns.

Note. Runtime performance is better if you specify the number of columns, rather than the number of rows.

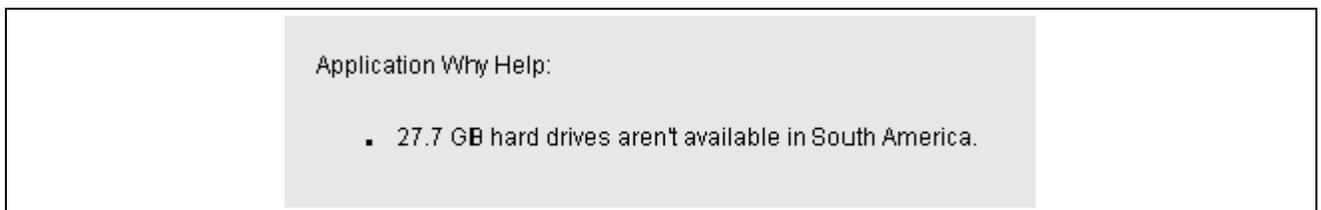
- **PARAM_BORDER:** The border attribute for the generated table. The default value is 0.
- **PARAM_CELLSPACING:** The cell spacing attribute for the generated table. The default value is 0.
- **PARAM_CELLPADDING:** The cell padding attribute for the generated table. The default value is 0.

See [Appendix C, “Advanced Configurator Form Controls,” Single-Select Image Table, page 458.](#)

Application Why Help

Use an Application Why Help template to represent a set of violation items of the current configuration in an unordered list format.

You can create the following application-level why help:



Application Why Help

The following code generates the form control in the figure:

```

<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_CAPTION, "Application Why Help:");
    generateApplicationWhyHelp(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->

```

See [Appendix C, “Advanced Configurator Form Controls,” Application Why Help, page 460.](#)

Form Control Why Help

Use a Form Control Why Help template to represent a set of violation items of a form control for the current configuration in the unordered list format.

You can create a Form Control Why Help like the example in the Application Why Help:

<p>Application Why Help:</p> <ul style="list-style-type: none"> ▪ 27.7 GB hard drives aren't available in South America.

Form Control Why Help

The following code generates the example form control:

```

<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "[MyComputerConfig].HDSelection");
    params.put(PARAM_CAPTION, "Control Why Help:");
    generateControlWhyHelp(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->

```

See [Appendix C, “Advanced Configurator Form Controls,” Form Control Why Help, page 460.](#)

Text Input Form Control

The text input template generates a text box for entering any text value. Unlike values that are entered into extern entry templates, text input values are not submitted to the engine. They are instead passed from page to page and loaded into the configuration object that is contained within the COP on each page. This allows any information that you store using the text input template to be included with standard configuration information, such as that acquired when you use the BOM item generator, which is saved when you run it against a compound model or integrate it with other applications.

Include the template for the extern entry text box in the following way:

```
<%@ include file="/calico/templates/html/TextInput.jsp" %>
```

For example, you can create the following text input control with an initial default value.

promotion code:

Text Input control

The following code generates the control in the example.

```
<!-- METADATA TYPE="CalicoControl" startspan -->
<%
if (params != null)
{

    params.clear();
    params.put(PARAM_OBJECTNAME, "TextInput_Ctrl01");
    params.put(PARAM_CAPTION, "Phone Number:");
    params.put(PARAM_TEXTINPUTDEFAULT, "212-555-1212");
    params.put(PARAM_ENTRYSIZE, "12");
    generateTextInput(params, request, out, session, cop.getConfiguration());
}
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

See [Chapter 27, “Using JSP Form Control Templates,” Parameters in the Inclusion Set, page 344.](#)

The two optional parameters are:

PARAM_TEXTINPUTDEFAULT: A default value that shows up when the text input box is initially shown.

PARAM_ENTRYSIZE: The width in characters of the text entry field. The default value is 15.

See [Appendix C, “Advanced Configurator Form Controls,” Text Input Form Control, page 461.](#)

Numeric Data Form Control

A numeric data control consists of a control caption or a caption image and a value of the numeric data that are displayed in the same line.

The include file that is required for the numeric data template is:

```
<%@ include file="/calico/templates/html/NumericData.jsp" %>
```

For example, you can create the following numeric data control for the decision point WattsSummation with the caption *Total Watts*.

Pick a Speaker >> Altec Lansing 2000 - Cambridge SoundWorks SP	Pick a Printer >> HP Laserjet - Epson 1500	Pick a Scanner - NEC Technologies PediScan >> Fujitsu ScanPartner 15C - Canon DR5080C
Pick a Hard Drive - 4.3 GB BATRA-33 - 8.4 GB BATA-33 >> 13.6 GB BATA-33 - Quantum 27.3 GB SCSI		Pick a Monitor - No Monitor - 15 Inch Monitor >> 17 Inch Monitor - 21 Inch Monitor
Submit Picks	Total Watts: 13.0	Clear Picks

Numeric data for watts after submitting picks

The following code generates the control in the figure:

```

<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "[_Application].WattsSummation");
    params.put(PARAM_CAPTION, "Total Watts");
    generateNumericData(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->

```

See [Appendix C, “Advanced Configurator Form Controls,” Numeric Data Form Control, page 461.](#)

Extern Entry

The extern entry template generates a text box for entering a value that is submitted to the Configurator engine using the `extern()` function.

Include the template for the extern entry text box in the following way:

```
<%@ include file="/calico/templates/html/ExternEntry.jsp" %>
```

For example, you can create the following extern entry control to enter a floating point value.

<input type="text" value="12"/> Discount %
--

Extern Entry control with a numeric value entered

The following code generates the previous control.

```

<!-- METADATA TYPE="CalicoControl" startspan -->
<%

```

```

if (params != null)
{

    params.clear();
    params.put(PARAM_OBJECTNAME, "[_Application].ExtVar1");
    params.put(PARAM_CONTROLID, "ExEn01");
    params.put(PARAM_CAPTION, "Enter Number Here");
    params.put(PARAM_FLOATENTRY, "true");
        params.put(PARAM_ENTRYSIZE, "3");
    generateExternEntry(params, request, out);
}
%>
<!-- METADATA TYPE="CalicoControl" endsan -->

```

See [Chapter 27, “Using JSP Form Control Templates,” Parameters in the Inclusion Set, page 344.](#)

The two optional parameters are:

- **PARAM_FLOATENTRY:** “True” processes the entered number as a floating point value. The default value “False” processes the number as an integer.
- **PARAM_ENTRYSIZE:** The width in characters of the text entry field. The default value is 3.

Using Configuration Form Control Templates

PeopleSoft Advanced Configurator supplies 11 form control templates. Each is a JSP page that generates HTML. Six of them enable the web user to select an item on an HTML form, using radio buttons, check boxes, drop-down lists, or selection lists. The other four form control templates generate output display information. One enables the web user to enter a text value using an edit box. Two of these form controls display constraint violations; one for the entire configuration session, and the other for its corresponding form control. The last display form control template generates a visible numeric value.

Three types of user-selectable form controls are available:

- **Group:** radio buttons and check boxes that are grouped as part of the selection point.
- **List:** drop-down or selection lists; one for each selection point.
- **Table:** Table form controls and Group form controls are alike, except that the Table form controls provide, in addition to the item name, columnar information for each form control item. The Table templates generate extra columns that are determined by the additional field descriptors in the model.

Each type of user-selectable form control can be either single-selectable or multiselectable. A multiselect form control enables the web client user to select more than one item for the selection point, whereas a single-select form control limits the user to a single selection for the selection point. A single-select list is drop-down, whereas a multiselect list is a list, typically scrollable. The following table itemizes the form control templates that are available in this release.

JSP Filename	HTML Implementation	Comment
SingleSelectGroup	<INPUT TYPE="radio">	Radio buttons
MultiSelectGroup	<INPUT TYPE="checkbox">	Check boxes

JSP Filename	HTML Implementation	Comment
SingleSelectList	<SELECT>	Drop-down list; single-selectable
MultiSelectList	<SELECT MULTIPLE>	List: multiselectable
SingleSelectTable	<INPUT TYPE="radio"> and <TABLE></TABLE>	Multiple columnar radio buttons
MultiSelectTable	<INPUT TYPE="checkbox"> and <TABLE></TABLE>	Multiple columnar check boxes
SingleSelectImage		An image for selecting a domain member
SingleSelectImageTable	 and <TABLE></TABLE>	A row or column of images for selecting among domain members
ApplicationWhyHelp		Violation messages for all form controls: application scope
ControlWhyHelp		Violation messages for an individual form control: form control scope
NumericData	caption text or image, and number text	Display number value aside caption or image
ExternEntry	<INPUT TYPE="text">	A text box for entering a value
TextInput	<INPUT TYPE="text">	A text box for entering a text value

Plugging Form Controls into the Application Pages

To use the form control templates, you need to be familiar with the following directories:

- Applications and processing JSP pages for the form control templates:

Solaris: `/opt/bea/weblogic81/config/CalicoDomain/
applications/CalicoApp/`

Windows NT: `\bea\weblogic81\config\CalicoDomain\
applications\Calico⇒App\`

- Configuration models:

```

Solaris:          /opt/bea/weblogic81/config/CalicoDomain/
                  applications=>/Calico=>
                  App=>
                  /Web-inf/models

Windows NT:      \bea\weblogic81\config\CalicoDomain\
                  applications\CalicoApp\Web->
                  =>
                  =>
                  =>
                  =>
                  =>
                  inf\models\

```

To use a form control template, your application pages need:

- A JSP `include` directive that includes the specified form control template.
- A specialized code-block that passes the required parameters to the specified form control template followed by a method call to generate and include it. This information is called the form control *inclusion set*.

Using a JSP Include Directive

The syntax to include a form control template, such as the `SingleSelectGroup` template, is:

```
<%@ include file="/calico/templates/html/SingleSelectGroup.jsp" %>
```

Place the include directives for the form control templates immediately following the include directive for the `CalicoStartFormInc.jsp` that is in each application JSP page.

For example:

```

<%@ include file="/calico/CalicoStartFormInc.jsp" %>
<%@ include file="/calico/templates/html/SingleSelectGroup.jsp" %>
<%@ include file="/calico/templates/html/MultiSelectTable.jsp" %>...

```

Note. Only one include directive `<% include file="templateFile" %>` is required for each type of form control template in the application JSP page. If you have five single-select list form control templates in an application JSP page, use only one JSP include directive to include them.

Using a Form Control Inclusion Set

To include a form control template using an inclusion set, bracket the inclusion set within these start and end tags:

```

<!-- METADATA TYPE="CalicoControl" startspan-->
  // set the parameters and include the path name for a form
  // control template here
<!-- METADATA TYPE="CalicoControl" endspan -->

```

Between the start and end tags, insert these required statements:

<code>params.clear();</code>	Initializes all parameters.
<code>params.put(parameter, value);</code>	Sets the required and optional parameters after <code>params.clear()</code> . The four required parameters for the interactive form controls are: <code>PARAM_OBJECTNAME</code> <code>PARAM_ATTRIBUTES</code> <code>PARAM_CAPTION</code> or <code>PARAM_CAPTIONIMAGE</code> <code>PARAM_CONTROLID</code>
<code>generateTemplateName(params, request, => out);</code>	Generates the included form control template file after the <code>params.put()</code> calls where you've set parameters. The <code>params.put()</code> statements can be in any order, provided that <code>params.clear()</code> precedes them and that <code>generateTemplateName()</code> follows them.

Example:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "VehicleSelection");
    params.put(PARAM_ATTRIBUTES, "Desc");
    params.put(PARAM_CAPTION, "Select a Vehicle");
    params.put(PARAM_CONTROLID, "VehicleSelectionControlID");
    generateSingleSelectGroup(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

Parameters in the Inclusion Set

At runtime, the parameters in the inclusion set are passed to the form control template, which uses them when it runs at the point where `generateTemplateName()` is called.

Any blank spaces between the values are ignored. For example, “Description, Part Number, Unit Cost” (with spaces) is the same as “Description,Part Number,Unit Cost” (without spaces).

Form control parameter values are specified as a string. The syntax for assigning multivalued parameters (in the application page) is a single string that uses a comma character delimiter between each value. Both the `PARAM_ATTRIBUTES` and `PARAM_COLUMNHEADINGS` parameter values may be multivalued.

These parameters are required for using the selection point form control templates:

Required parameter	Comments
PARAM_OBJECTNAME	The object name is the unique object name (“<modelnameConfig>.decisionpointname”), which is used to retrieve the items of a form control.
PARAM_CONTROLID	The control ID is the unique identifier for the HTML form control. The application needs a unique identifier for each form control other than the object name because multiple controls might map to the same decision point. This identifier is used for client-side scripting and server-side processing.
PARAM_ATTRIBUTES	<p>The attributes are the data attributes for the selection point that are to be displayed on the form control in the UI. Only Table form controls can use <i>multiple</i> display attributes from the model.</p> <p>For internal domain members that are specified in the if PARAM_ATTRIBUTES parameter, corresponding attributes must be defined in the model for the domain members.</p> <p>For external domain members, if the attribute name (column names of the table that stores the description for the domain members) is ambiguous (more than one table has the same column name), you must specify its fully qualified database name—tablename.columnname.</p> <pre>params.put (PARAM_ATTRIBUTES, <attribute name, => attribute name ...>);</pre> <p>OR</p> <pre>params.put (PARAM_ATTRIBUTES, <tablename.columnnam=> => => => => => e, tablename.columnname ...>)</pre> <p>Setting the PARAM_ATTRIBUTES parameter in your inclusion set is optional; however, for deployment you should set it so that the attributes of the model are displayed. If PARAM_ATTRIBUTES is not explicitly set, the default display will be the domain member names, which may be useful during development in that you do not need to add attributes to test-case models.</p>
PARAM_CAPTION	The caption is the string to be displayed as the form control caption. This parameter is applicable to all the form control templates.
PARAM_CAPTIONIMAGE	<p>The caption Image is the image to be displayed as the form control caption. This parameter is applicable to all the form control templates</p> <p>Specify either a caption or a caption image, but not both.</p>

The following table lists optional parameters for the form control templates:

Optional parameter	Comments
PARAM_SORT	The sort parameter specifies whether to sort form control data based on the form control states; <i>true</i> and <i>false</i> are the valid values.
PARAM_FILTERELIMINATEDITEMS	<p>The filter-eliminated items parameter renders all eliminated domain members in the model unavailable to the form controls in the application page. If the parameter is set to <i>true</i>, the eliminated domain members in the model are not available to the form control. If it is unspecified or set to <i>false</i>, the eliminated domain members in the model are available to the form control.</p> <p>You can restrict the filtering to outside a range of elimination levels that are captured between a lower and an upper elimination level that is specified by the parameters PARAM_ELIMINATIONLEVEL_LOWER and PARAM_ELIMINATIONLEVEL_UPPER.</p> <p>Therefore, if PARAM_ELIMINATIONLEVEL_LOWER is 4, and PARAM_ELIMINATIONLEVEL_UPPER is 7, then the eliminated domain members with elimination levels greater than 3 and less than 8 are available to the form control.</p>
PARAM_ELIMINATIONLEVEL_LOWER	The lower elimination level is used to specify the lower bound of the elimination level range for form control items.
PARAM_ELIMINATIONLEVEL_UPPER	<p>The upper elimination level is used to specify the upper bound of the elimination level range for form control items.</p> <p>Note. To specify the range for elimination levels, you must set <i>both</i> parameters, PARAM_ELIMINATIONLEVEL_LOWER and PARAM_ELIMINATIONLEVEL_UPPER.</p>
PARAM_COMPARATOR	Specifies a Java class name to be used with a custom comparator for sorting control items. The class that is specified must implement the java.util.Comparator interface. It passes a custom-written Comparator object to the ControlData.iterator method. The iterator method then sorts domain members using Comparator before returning the iterator to the front-end. Sort can be performed on quantity or any other attribute from the model.
PARAM_COLUMNHEADINGS	The column headings are the labels that the Table form control displays for the columns of domain member attributes.
PARAM_GENERATENONEMODE	<p>The Generate None mode is the position for the None item (explicitly selecting none of the picks). This option is available only for single-select controls. The corresponding decision point must be optional. Available options are:</p> <p>0: Do not generate <i>none</i> value.</p> <p>1: Generate <i>none</i> value before other options.</p> <p>2: Generate <i>none</i> value after other options</p>
PARAM_GENERATENONETEXT	The Generate None text is the string to be displayed for the None item. This option is available only for single-select controls. The corresponding decision point must be optional.

Optional parameter	Comments
PARAM_EVENTHANDER	<p>The event handler is the JavaScript event handler for the standard JavaScript events of form control items.</p> <p>Put your JavaScript event-handler function in the body of the application pages, outside of the inclusion set.</p> <p>For the event handlers, you must write your own JavaScript function to handle the events, such as <code>onClick</code> or <code>onBlur</code>, and then set the parameter <code>PARAM_EVENTHANDER</code> to that function name. Otherwise, you can use the standard event handlers, such as:</p> <pre>this.<form-name>.submit()</pre> <p>Syntax:</p> <pre>params.put(PARAM_EVENTHANDER, "onClick=> 'this.<form-name>.submit()'</pre> <p>" or</p> <pre>params.put(PARAM_EVENTHANDER, "onChange='my=> JavascriptSubmit()'"</pre> <p>Note. Common functions such as <code>onClick</code>, <code>onChange</code>, and <code>onFocus</code> are JavaScript events. <code>onClick</code> is an event for Group and Table form controls, but you can use different JavaScript events. <code>onChange</code> typically is used for List form controls, but you can use other JavaScript built-ins such as <code>onFocus</code> and <code>onBlur</code>.</p>
PARAM_CONTROLSIZE	<p>The control size is the number of visible items for the HTML form control. This parameter applies only to the List form controls.</p>
PARAM_GENERATEFIRSTITEMTEXT	<p>You can add an item with any description (string type) to the beginning of a single-select list control. For example, you can add "Select a hard drive" to be displayed as the top element in the drop-down list. When the user selects an item from the drop-down list, (for example, "12 GB Hard Drive"), the first item entry ("Select a hard drive") then disappears.</p>

Specifying the Model and Locale Properties for the Solution

The user interface (UI) properties for a solution are contained in the configuration file, `CalicoUI.properties`. Each solution, or version of a solution, requires a separate copy of `CalicoUI.properties` to be modified and maintained for that solution only. It must be placed in the application folder that houses your application JSP pages.

`CalicoUI.properties` contains mainly display properties, which are optionally used by the application. However, four model properties and two locale properties exist that the form control templates need and that you must specify in the solution `CalicoUI.properties` file:

- `calico.ui.version`: A unique number for each application that is hosted by the same application server; one entry per `CalicoUI.properties` file.
- `calico.model.name`: Holds the model name. Valid model names are any subfolder name under `<install_dir>/PeopleSoft Applications/Configurator/`, which houses the models. The modeler sets the model name when it is created.

- `calico.model.version`: Gets the latest compiled model with the specified version number.
- `calico.model.compileversion`: The generated ID for a compiled model, for example, `20010221-161729-588`.
- `calico.locale.language=en`: Indicates which set of text in the XX file that provides text for captions and controls in the desired language.
- `calico.locale.country=US`: Indicates which country variant of the locale language that is specified. If a custom language variant is used, the `calico.locale.variant=` property is provided for specification instead.

To specify model and locale properties:

1. If you haven't already done so, transfer a copy of `CalicoUI.properties` from `<install directory>\weblogic81\config\CalicoDomain\applications\CalicoApp\solutions\ComponentSample` to the application folder where the JSP pages are kept.
2. Call the property values from each form control that uses the display properties:
 - If it is a template form control, insert the property as a parameter in its JSP inclusion set. For example:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, "[SmallBusTelecom].ServicePlanSelection");
    params.put(PARAM_ATTRIBUTES, "Desc");
    params.put(PARAM_CAPTION, "Select a Service Plan");
    params.put(PARAM_CONTROLID, "ServPlanSelectionControlID");
    params.put(calico.ui.version, "SBQ01.04.01");
    generateSingleSelectGroup(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

- If it is a custom form control, fetch the property values with calls from your code.
3. If the language to use in the UI is other than the United States variant of English, specify which by inserting the code for the desired language and variant. If you have a custom language variant set, specify the desired language code in `calico.locale.country=en`, comment out `calico.locale.country=US`, uncomment `calico.locale.variant=`, and insert in it the correct code.

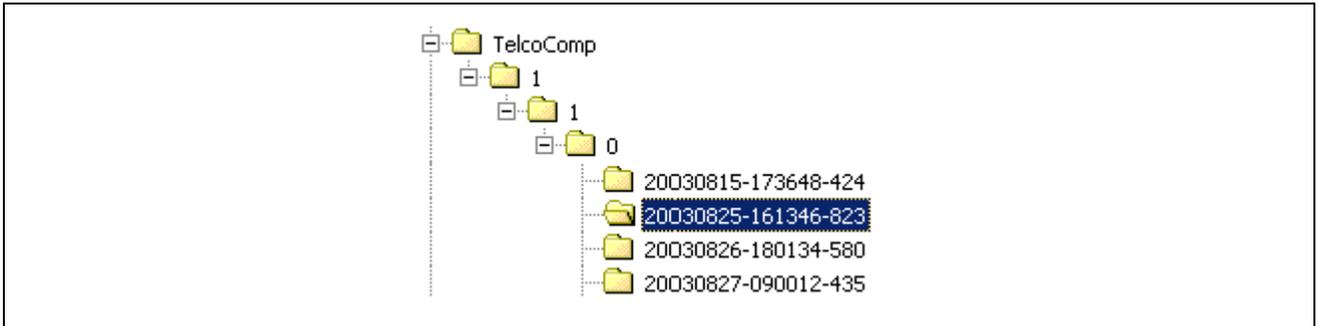
Assigning a Specific Model Version to Use for Configuration

If you use only the major number and the minor number of the model version for setting the `calico.model.version` property (such as `calico.model.version=0-1`), then by default, the form control templates use the latest compiled version from among all models within the `subversions` folder under version 0.1.

The model name and version (major-minor) are set in the model by the modeler. By changing these in the `CalicoUI.properties` file, you tell the application which model to use rather than the default latest compiled version. The compiler assigns the micro (sub-version) and compile versions. The WebLogic server by default loads only the most recent version (major-minor-micro) and compile version of every model.

To assign a specific model version to use:

1. Set the `calico.model.version` property to the complete version number. For example, `calico.model.version=0-1-4` represents version 0.1.4.
2. Uncomment the `calico.model.compileversion` property in `CalicoUI.properties`.
3. Set the `calico.model.compileversion` property to the name of the folder that houses the compiled model. For example, using the following directory tree for Configurator models, you would set `calico.model.compiledversion` property to `200030825-161346-823`.



Setting the compile version (optional)

Specifying Solution Information Properties

You use solution information property values to generate a Solution List for Configurator client application. A solution is the collection of pages, images, model files, and supporting files that make up the applications that solve a business problem, such as online configuration and order management. The solution list is the filenames, structures, and version information for that solution.

The solution list could also be used in a server-based deployment.

Solution properties are:

calico.solution.name	Name of the solution. This acts as the key in the solution list. By default (if this property isn't specified), the name is the name of the directory that the solution resides in (the value of <code>calico.solution.root</code> , which is not specified here). Default is the name of the directory in which the solution resides.
calico.solution.version	The version of the solution. This helps identify different versions (revisions) of the same solution, but isn't used in this version of Configurator. Default is <code>calico.solution.root</code> or as specified in the properties file.
calico.solution.description	The text to be displayed in the solution list as a link to this application. The default is <code>calico.solution.name</code> or as specified in properties file.
calico.solution.restorePolicy	The restore policy to use when you launch stored configurations. The policy determines which version of the model to use in launching the configuration in the web application. The default is <code>1</code> or as specified in the properties file.
calico.page.start	The page to redirect to when starting a new configuration.
calico.page.restore	The page to redirect to when restoring an existing configuration. If <code>calico.page.restore</code> is blank, the restore goes to the page that is specified in <code>calico.page.start</code> .

If these properties aren't specified, the `SolutionInfo` class assigns default values.

Specifying Display Properties

The names of the form control property types correspond to the names of the display attributes, as shown in the following list:

Display property types in CalicoUI.properties file	Applicable form control template types
calico.control.tag.<property>	List
calico.control.image.<property>	Group and Table
calico.control.textcolor.<property>	Group and Table
calico.control.captioncolor.<property>	List, Group, and Table

PeopleSoft Advanced Configurator provides the following set of properties to indicate the state of an item in a control:

- Selectable
- User-selected
- Computer-selected
- Default-selected
- User-eliminated
- Computer-eliminated
- Conflict

These properties cause the display of designated text symbols, images, or both that are beside the items in a control to indicate to the user their availability for selection.

You can use the item state properties as examples for creating your own properties or modify them to meet your needs.

Displaying Delta Information

Using the COP XML interface, Advanced Configurator generates two types of delta configuration information at runtime:

- Structural

Structural deltas include component additions, deletions and changes, and connection additions, deletions, and moves. Structural and component delta information applies to configurations that are generated from compound models.

- Component

Component information deltas indicate changes between two configurations for a single component and apply to the components in both component and compound configurations. The information can include:

- Added, changed, and deleted configuration attributes.
- Added, changed, and deleted choices.

Advanced Configurator returns delta information using an XML interface. The existing ConfigDetails request in the messaging interface (COPXML) can include requests for delta information. By default, the deltas that are returned represent the latest set of saved changes or those from a specified date range. In addition, the available component delta information will be enhanced to include changed expression values. Peoplesoft Enterprise CRM 9 Order Capture and Service Management use delta information in transactions.

See Also

[Chapter 16, “Retrieving Configuration Information,” page 201](#)

Displaying Delta Pricing

You can modify the pricing display for either all form control items or for the form control items belonging to specified selection points. To display delta prices, you must first enable delta pricing in the application and then specify how it is displayed. Enabling delta pricing occurs in initial implementation of `jspInit()` in the body of your application pages.

1. Enable delta pricing:

To enable delta pricing for all form controls, use this statement:

```
setPricing(true);
```

This statement also enables total pricing for the entire configuration.

To enable delta pricing for *specific* form controls, use this statement:

```
setPricingControls(); // accepts as arguments a map of the
                    // decision points' object names (strings)
                    // and price attributes (strings).
```

Pass to `setPricingControls()` the object names that you specify in the `PARAM_OBJECTNAME` parameters of the form controls, along with the price attribute for that object.

Note. For better performance, set the pricing variables rather than calling the corresponding set methods.

2. Set display properties.

The basic format is:

```
calico.pricing.add=[+ {0}]
calico.pricing.subtract=[- {0}]
```

The values that are shown result in a display string that appears alongside the selection. If the delta price is an additional 20.00 USD, the delta price appears in the following way:

[+ \$20.00]

By replacing the characters, you can change the display. For example, if you replace the square brackets with angle brackets, and + with Add, the display becomes `<Add $20.00>` for a positive delta price. Replacing - with Subtract gives `[Subtract $20.00]` for negative deltas.

Sample code:

```
public void jspInit()
{
    setPricing(true);
}
```

```

Map pricingControls = new HashMap();
pricingControls.put("[BMWConfig].SeriesSelection", "UnitPrice");
pricingControls.put("[BMWConfig].OptionsSelection", "UnitPrice");
pricingControls.put("[BMWConfig].EngineSelection", "basePrice");

setPricingControls(pricingControls);
}

```

Application Page Example

The following code provides a complete example of an application JSP page.

```

<HTML>
<HEAD>
  <TITLE>Sample Page</TITLE>
</HEAD>
<BODY>

<%!
  public void jspInit()

  // turn on pricing for controls
  setPricing(true);

  Map pricingControls = new HashMap();
  pricingControls.put("[myPCConfig].ProcessorSelection", "UnitPrice");

  setPricingControls(pricingControls);
  }
%>
<FORM name="TestForm" method="POST" action="CalicoProcessForm.jsp">
<%@ include file="/calico/CalicoStartFormInc.jsp" %>
<%@ include file="/calico/templates/html/SingleSelectGroup.jsp" %>

<!-- Inclusion set starts below -->
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME,
      "[myPCconfig].ProcessorSelection");
    params.put(PARAM_ATTRIBUTES, "Desc");
    params.put(PARAM_CAPTION,
      "[myPCConfig].ProcessorSelection");
    params.put(PARAM_CONTROLID, "ProcessorSelectionCtrl");
    generateSingleSelectGroup(params, request, out);
  }

```

```
%>
<!-- METADATA TYPE="CalicoControl" endsan -->

<%@ include file="/calico/CalicoEndFormInc.jsp" %>
</FORM>
</BODY>
</HTML>
```

Configuring a Form Control Template

You must use `getControlData()` in configured form control template JSP pages, as in the following Java statement:

```
ControlData varName = getControlData(objectName, request, params);
```

Application Scope

You can configure HTML-level display properties, such as the text color of the form controls, by modifying the properties in the `CalicoUI.properties` file. The form control templates load these properties upon their initialization.

Template Scope

You can configure how the display data is altered depending on the data for the model. You can configure the behavior of the display to:

- Display domain member attribute information.
- Sort the form control items.
- Eliminate form control items of eliminated domain members.
- Eliminate the form control items of the eliminated domain members whose elimination levels are beyond an inclusionary range.
- Display delta pricing and the total price.

To configure the state of an individual instance of a form control:

1. Make a copy of the type of form control template that you want, rename the copy, and put that copy in your application directory.

When you use the JSP include directive to include your template, be sure to use the path to your application directory.

Note. Do not modify any of the form control template JSP files: they are installed as read-only files.

2. Modify its JSP code.
3. Include your custom form control template in any of your application JSP pages whenever you use that type of form control.

Loading the Form Control Data from the Model

The form control templates get their corresponding selection point (decision point) data from the model with the following JSP template code:

```
String objectName = (String)params.get(PARAM_OBJECTNAME);
ControlData ctrlData = getControlData(objectName, request, params);
```

Loading the UI Properties for a Control

The form control templates load the specified UI properties (from the CalicoUI.properties file) with the following JSP template code:

```
// set up display properties
String[] stateMap = (String[])request.getAttribute(STATEMAP);
String[] tagMap = (String[])request.getAttribute(TAGMAP);
String[] imgMap = (String[])request.getAttribute(IMAGEMAP);
String[] textColorMap = (String[])request.getAttribute(TEXTCOLORMAP);
Properties props = (Properties)request.getAttribute(UIPROPS);
```

The form control templates set their properties with the following JSP template code:

```
// set up the display properties for control items
state = (int)ctrlItem.getFlags();
stateFlags = stateMap[state];
tagItemState = tagMap[state];
imgItemState = imgMap[state];
colorItemText = textColorMap[state];
```

The names of the form control property types correspond to the names of the display attributes, as shown in the following list:

Display property types in CalicoUI.properties file	Applicable form control template types
calico.control.tag.<property>	List
calico.control.image.<property>	Group and Table
calico.control.textcolor.<property>	Group and Table
calico.control.captioncolor.<property>	List, Group, and Table

Registering Custom Form Control Templates

The addCtrl() function should be called within any configured form control template. addCtrl() registers a Configurator form control. It adds each control along with its selection point name to a form-hidden attribute called CalicoCtrlMap, which is generated by CalicoEndFormInc.jsp for each Configurator page. This is necessary so that the Configurator JSP processor page (CalicoProcessFormInc.jsp) knows which form controls need to be processed and which to ignore (non-Configurator form controls).

addCtrl() is used by most of the Configurator form control templates.

Custom Form Control Template Example

The purpose of providing the sample template is to explain how developers could configure the behavior of a Configurator control by displaying eliminated control items differently based on their elimination levels returned from COP:

If the elimination level for an eliminated control item is ≥ 0 and ≤ 4 , the item is hidden (not shown).

If the elimination level for an eliminated control item is ≥ 5 and ≤ 9 , the item is displayed as unavailable with the X image.

If the elimination level for an eliminated control item is ≥ 10 , the item is displayed as normal (without being unavailable for entry or the X image).

In this example template, we use 0, 4, 5, 9, 10 and so on for the lower and upper bounds to display control items differently. Developers can define a range of values for the elimination level that they want to handle in the template that they create because these values depend on the elimination levels that are defined for the constraints by the modelers.

In the following figures, after the Prima Base Celeron 43 Mhz MiniTower pick is submitted, the control on the right has only one item left and is shown as user-selected (finger icon). All other items in the control have elimination levels that are ≥ 0 and ≤ 4 . Therefore, they are not shown on the page.

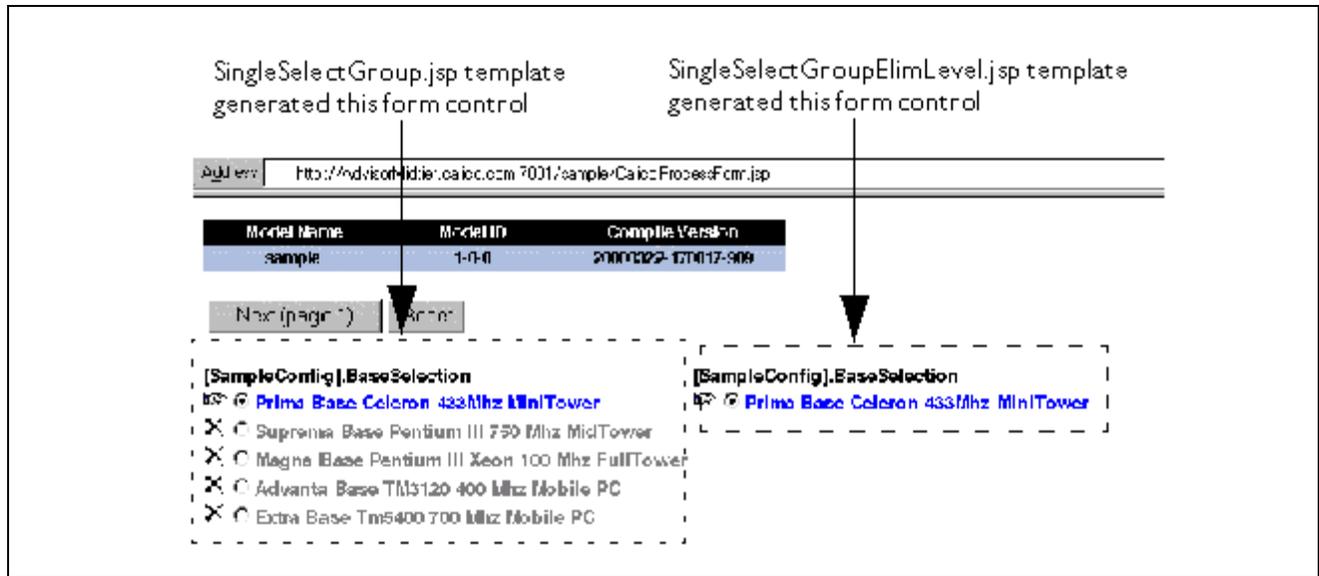
The following web page illustrates using the same model decision point: the top left form control with the SingleSelectGroup.jsp template, and the top right form control with the SingleSelectGroupElimLevel.jsp template:

Model Name	Model ID	Compile Version
sample	1-0-0	20000322-1700-17-909

Next (page 1) Reset

Web page

The next figure illustrates the differing effects of using the standard SingleSelectGroup.jsp form control template and of using a configured (sample) SingleSelectGroupElimLevel.jsp form control template. The top left form control was generated by the SingleSelectGroup.jsp template, and the top right form control was generated by the SingleSelectGroupElimLevel.jsp template:



Second page

Common Errors

The following list describes common errors for those who implement the web applications using Configurator JSP pages:

- `calico.model.compileversion`: The `calico.model.compileversion` entry of the `CalicoUI.properties` file is used to specify the compiled version for the model to be loaded for the JSP page.

By default, this entry is commented out and the most recent compiled version is loaded for a model. Don't uncomment this entry unless a specific compiled version of a model is given.

- Better performance: Add `<%@ page session="false" %>` to a JSP page if the page does not reference the session-implicit variable. This should apply to the `CalicoProcessForm.jsp` file as well.
- `CalicoEndFormInc.jsp`: A Configurator JSP page must include both `CalicoStartFormInc.jsp` and `CalicoEndFormInc.jsp`. For example:

```
<%@ include file="/calico/CalicoStartFormInc.jsp" %>
.....
<%@ include file="/calico/CalicoEndFormInc.jsp" %>
```

`CalicoEndFormInc.jsp` calls the `release()` method for the COP. If a system quickly runs out of memory when running Configurator JSP pages, verify that `CalicoEndFormInc.jsp` is included properly on each page.

- Variables declared within JSP `<%! %>` blocks: Be careful using variables that are declared within `<%! %>` blocks of the JSP pages. For example:

```
<%!
  HashMap myMap = new HashMap();
  ...
%>
```

The values of the variables that are declared within the `<% ! %>` blocks are accessed and modified by more than one user when multiple clients access the pages concurrently unless the access to such objects is properly synchronized. If such values are meant to be user-specific, the preceding variables that are used by other functions or codes may easily run into problems.

- **Template files:** The name of the control-generating function should be changed as well as in any custom templates (for example, `generateMyEnterpriseLargeSingleSelectTable(params, request, out)`, `generateMyEnterpriseSmallSingleSelectTable(params, request, out)`) so that no confusion results if multiple templates of the same control type exist on one page.
- Keep the custom templates in the application directory instead of in the `calico` directory.
You should include only template files that are used on the JSP page.
- `StringBuffer.append()`: Use `StringBuffer.append()` for string concatenation instead of using the `+` operator of the `String` class.
- Avoid unnecessary evaluations and object creations in Java codes, especially codes within loops—for example, `while` and `for` loops.

CHAPTER 28

Using the Page Editor Extensions for Dreamweaver

This chapter provides an overview of Dreamweaver extensions and discusses how to:

- Edit CalicoUI.properties.
- Insert a Configurator runtime object.
- Edit properties of Advanced Configurator objects.

Understanding Dreamweaver Extensions

PeopleSoft Advanced Configurator provides extensions for Macromedia® Dreamweaver™ that help the web developer build a website for a Configurator model. This chapter describes the Configurator extensions and how to use them. If you do not use Dreamweaver, skip this chapter.

See [Chapter 27, “Using JSP Form Control Templates,” page 325](#).

Configurator extensions for Dreamweaver include:

- A Configurator item in the Commands menu of the taskbar of the Dreamweaver Document window.

This menu item tells you which version of the extensions you have, enables you to edit the CalicoUI.properties file, and add it to your website.

- A Configurator panel in the Dreamweaver Objects palette.

This panel enables you to insert Configurator objects into the active Document window.

- A Configurator item in the Insert menu of the task bar of the Dreamweaver Document window.

This menu item also enables you to insert Configurator objects into the active Document window.

- Dreamweaver Property inspectors for most of the Configurator objects.

These inspectors enable you to edit properties of Configurator objects that are in the active Document window.

Advanced Configurator Runtime Objects

A Configurator object is a graphical user interface that creates JavaServer Pages (JSP) syntax on a JSP page when you insert the object into the active document window in Dreamweaver. You can view the syntax created by an object by having the Dreamweaver HTML Source window open as you insert objects or edit their properties.

See [Chapter 27, “Using JSP Form Control Templates,” page 325](#).

There are 10 Configurator objects that you can insert into the active Document window. Two of the objects are general purpose:

- Form
- Button

You need a Configurator Form to process other Configurator objects. A Configurator Button submits a page, or resets the picks on a page.

Note. You must use the form and button created by the Configurator Form and Configurator Button objects. They set attributes to work with your Configurator model.

Do not use more than one Configurator form on a JavaServer Page.

Three Configurator objects create form controls for selection points in the model:

- List
- Group
- Table

In fact, each of these three objects creates two types of form controls:

- Single-select: Enables the user to select only one item. A drop-down list box and radio-button group are single-select form controls.
- Multi-Select: Enables the user to select one or more items. A text list and check box group are multi-select form controls.

The type of selection point—whether it is optional or required—determines which type of form control these three Configurator objects create.

Note. The modeler creates the selection points in your model. A selection point is that point in your model where the user selects one or more items from two or more items. A selection point is optional if the user can select none of the items, and required if the user must select at least one item.

Two Configurator objects create a single-select form control that uses images to select an item:

- Single-select image control
- Image table

Single-Select Image Control places an image on the page where you indicate, to select one domain member of a selection point. Image Table places images for every domain member of a selection point in either a row or a column on the page, to select any domain member of a selection point.

Two Configurator objects create messages for constraint violations:

- Control Why Help
- Application Why Help

Control Why Help creates messages for a selection point. Application Why Help creates messages for the entire model.

Configurator NumericData is the tenth object. It displays the total—that is, the summation—for a numeric attribute of user picks, such as price, at runtime.

Creating a Solution

To create a Configurator web application in Dreamweaver, do the following:

1. In Windows Explorer, create a directory for your website files; for example:

```
D:\Dev\WebSites\myWebApp1
```

2. In Dreamweaver, create a site (select Site, New Site).
3. Edit the CalicoUI.properties file.

See [Chapter 28, “Using the Page Editor Extensions for Dreamweaver,” Editing CalicoUI.properties, page 361](#).

4. Create JSP pages by inserting Configurator objects into an active Document.

See [Chapter 28, “Using the Page Editor Extensions for Dreamweaver,” Editing Properties of Advanced Configurator Objects, page 374](#) and [Chapter 28, “Using the Page Editor Extensions for Dreamweaver,” Inserting a Configurator Runtime Object, page 362](#).

See Also

[Chapter 23, “Understanding the Runtime System,” Deployment for a Web Application Based on a Single Component Model, page 302](#)

[Part 8, “Advanced Configurator System Administration,” page 411](#)

Editing CalicoUI.properties

The CalicoUI.properties file must be in your web application directory with your JSP pages. Add it after you create your site in Dreamweaver. This file determines the display of the following Configurator objects in your application:

- List
- Group
- Table
- Numeric Data

Before you add the CalicoUI.properties file to your web application directory, you must first edit it to apply to your model.

To edit and add the CalicoUI.properties file to your web application directory, from an active Document window:

1. Select Commands, Edit Calico UI Properties File.

A dialog box asks if you want to add the Configurator UI properties file to your site.

2. Click OK.

The Add Calico UI Properties File To Site dialog box appears, so that you can edit the file before adding it to your application directory.

3. Set values for the following:

Model Information File	Browse to the model information file, and click it, or enter its full path and name.
	<hr/> Note. The model information file is created when the model is created and compiled using the PeopleSoft Visual Modeler. Obtain this file from the modeler, or locate it in the appropriate Visual Modeler directory; for example, D:\Configurator\ViM\Audio\Audio.modelinfo.xml. <hr/>
Model Name and Model Version Number	If you browsed to the model information file, the dialog box automatically fills these fields, reading from the file. Otherwise, enter the model name and version number.
UI Version Number (user interface version number)	Enter a unique number for your application. See Part 8, “Advanced Configurator System Administration,” page 411 .

Inserting a Configurator Runtime Object

To insert a Configurator object into the active Document window:

1. Select Window, Objects.
The Dreamweaver Objects palette appears.
2. Click the down arrow in the upper right-hand corner of the Objects palette, and in the menu that appears, click Advisor.
The Advisor panel appears.
3. Drag and drop the object that you want to insert from the Advisor panel.
The object’s insert dialog box appears.

Note. You can also insert Configurator objects from the Configurator item of the Insert menu.

4. Complete at least the required fields of the insert dialog box, and click OK.
For most of the Configurator objects, if you have not previously included CalicoProcessForm.jsp in your web application directory, a dialog box prompts you whether to include it.
5. Click OK.
CalicoProcessForm.jsp processes the Configurator form on your JSP page before CalicoProcessFormInc.jsp processes it in the normal way. By default, CalicoProcessForm.jsp does nothing, and adding it to your website has no effect. Modify this file only if you want to preprocess your Configurator form.

See [Chapter 27, “Using JSP Form Control Templates,” page 325](#).

Inserting a Form

You need a Configurator form to process all of the other Configurator objects on the JSP page. All other Configurator objects on the page must be inserted in the same Configurator form.

You insert the Configurator Form object into the active Document window from the Advisor panel of the Dreamweaver Objects palette.

If you try to insert another Configurator object before you insert the Configurator form, a message prompts you to insert the form with the other object.

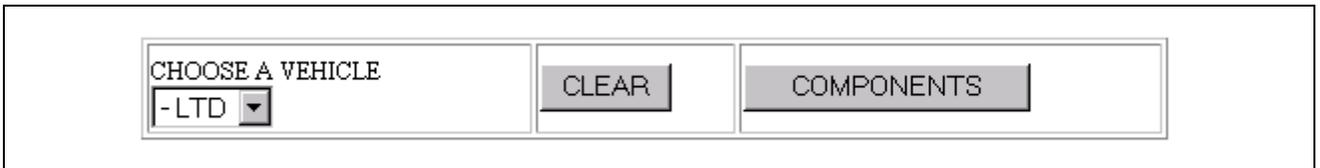
Warning! Do not use more than one Configurator form on a JavaServer Page.

Inserting a Button

A Configurator button does one of the following:

- Submits the page.
- Resets the page.

The submit button redirects to the URL that you set. The reset button sets picks on the page to the state that they had when the page was generated, and can redirect to another URL, if you want.



Submit and reset buttons

CLEAR is a reset button. It sets the user's choice of a vehicle to what it was when the page was generated. COMPONENTS redirects to the URL of the page for choosing components.

Insert a Configurator Button object into the active document window from the Advisor panel of the Dreamweaver Objects palette.

Set the properties of a Configurator Button object in the Insert Calico Button dialog box.

Label	Enter a label for the button, such as CLEAR for a reset button, and GO TO PAGE 2 for a submit button.
Button Type	Click either Submit or Reset.
URL	Browse to the page that you want to redirect to, and click it. You need not set the URL for a reset button, if you want to remain on the same page.

Inserting a List

A Configurator List is one of the following:

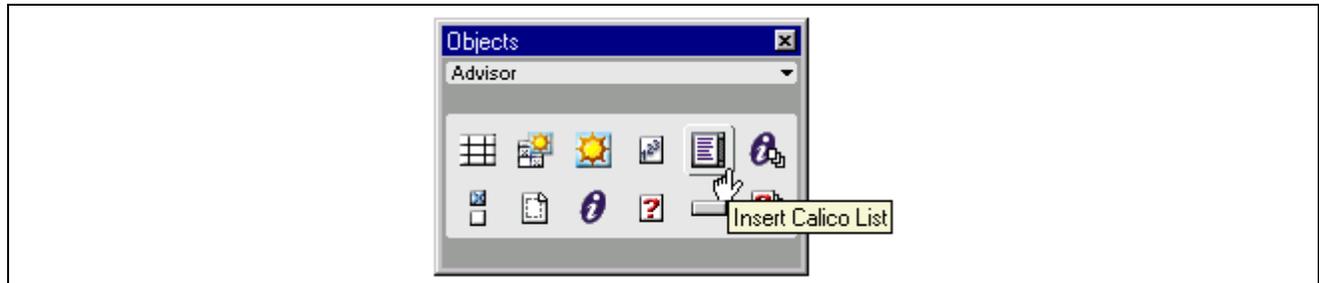
- A drop-down list box for a single-select selection point.
- A text list for a multi-select selection point.

The drop-down list box initially displays only one item, but expands to display all of the items available for a selection point. The user can select only one item. The text list displays the number of items that you set, and scrolls through additional items, if any. The user can select one or more items.



A drop-down list and two text lists

PICK A VEHICLE is a drop-down list box of four items. The PICK COMPONENTS text list is set to display one item, and scrolls through three others. The PICK KITS text list is set to display three items. Because the selection point has only three items, the list has no scroll bars. Insert a Configurator List object into the active document window from the Advisor panel of the Dreamweaver Objects palette.



Inserting a Configurator List from the Advisor panel

Set the properties of a Configurator List object in the Insert Calico List dialog box:

- | | |
|----------------------------|--|
| ID | Enter an ID that is unique across all controls for your application. |
| Selection Point | Browse to the <ModelName>.modelinfo.xml file, click it, and choose one of the selection points in the drop-down list box. (The model information file is created when the model is created and compiled.) |
| Caption Type | Select either <i>text</i> or <i>image</i> from the drop-down list box. |
| String or Path | If you want the caption to be text, enter it. If you want the caption to be an image, browse to the image, and click it. |
| Control Size | Enter the number of items that you want to display. The user must scroll to see more items. (This field applies to only multi-select lists.) |
| Sort | Click to order picked items first in the list after a pick is submitted. |
| Show Eliminated | Click to display elimination levels, then enter the upper and lower limits, inclusive, for the elimination levels that you want to display. (The modeler defines elimination levels in the model.) |
| Show “None” | Select one of the items in the drop-down list box. First places the None item at the top of the list. Last places the None item at the bottom of the list. Do Not Show excludes a None item from the list. (This field applies only to single-select lists.) |
| Replace “None” With | Enter text to identify the item in the list that lets the user choose nothing. “None” is used if you leave this field blank. |
| First Item Text | For a single-select list, enter the name of the control item that you want to appear first in the list. |
| Events | Click, then enter the HTML syntax to handle an event for this control. For example: |

```
onBlur = "foo()"; onClick= "foo2()"
```

Attributes

Click, then enter the attributes that you want to include. Separate multiple attributes with commas. Attributes replace domain member names. To display attributes, the modeler should create an attribute for domain member names; for example, an attribute called “DomMbrName” or “Desc.”

Note. If you want column headings for your attributes, you must use the Configurator Table object, which creates a radio-box group or check box group, with attributes in columns, and column headings.

Inserting a Group

A Configurator Group is one of the following:

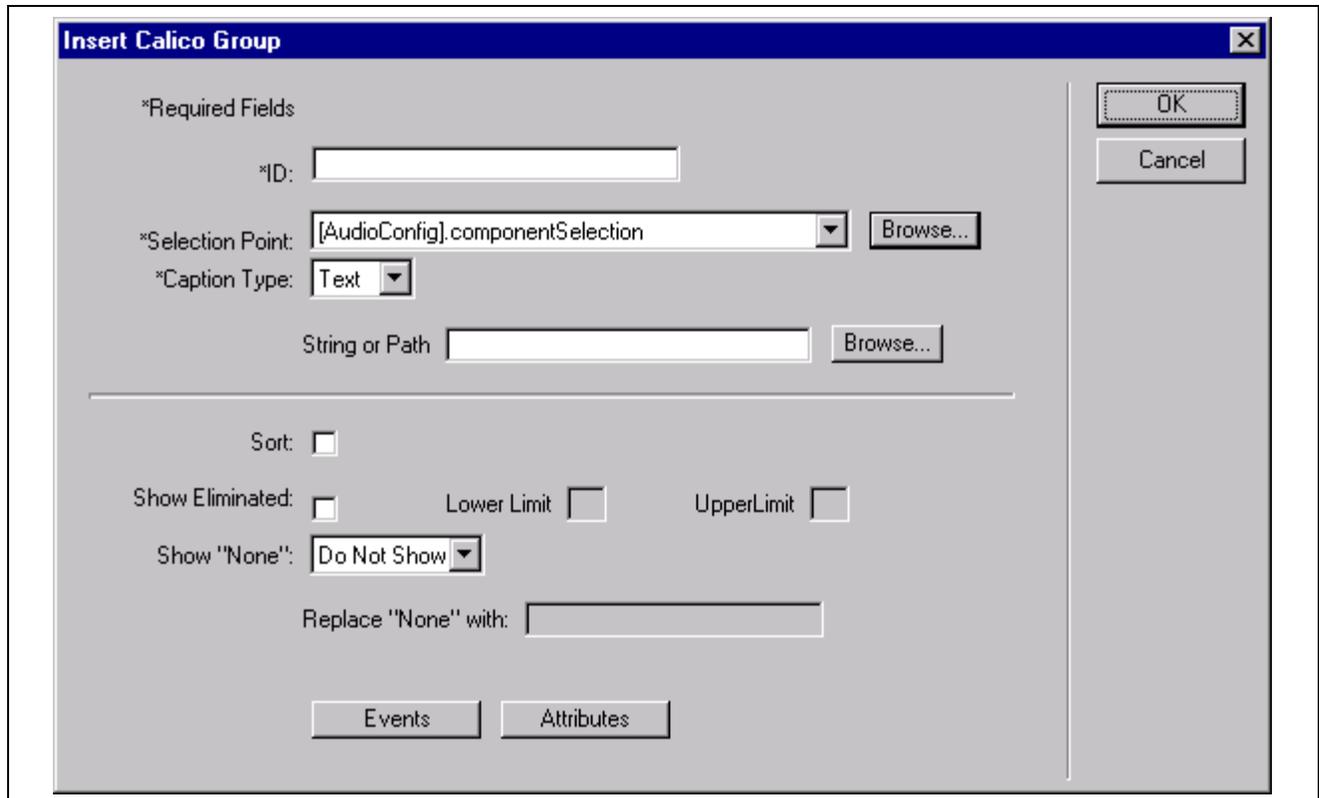
- A radio-button group for a single-select selection point.
- A check box group for a multi-select selection point.

With radio buttons, the user can select only one item. However, you can include an item at the top or bottom of the group to enable the user to select nothing, if the selection point is optional. With check boxes, the user can select one or more items, and can select nothing by clearing all of the check boxes.

SELECT A VEHICLE	SELECT COMPONENTS	SELECT KITS
<input type="radio"/> LTD	<input type="checkbox"/> RadioTapeCDSingle	<input type="checkbox"/> DashSpacer
<input type="radio"/> F150	<input type="checkbox"/> RadioTape	<input type="checkbox"/> Trunk
<input type="radio"/> XJ6	<input type="checkbox"/> RadioTapeCDmultiple	<input type="checkbox"/> UnderSeat
<input type="radio"/> Z3	<input type="checkbox"/> CDchanger	

A radio-button group and two check box groups

SELECT A VEHICLE enables the user to select only one vehicle. If the selection point is optional, you can include a None button at the top or bottom of the group to enable the user to select nothing. SELECT COMPONENTS and SELECT KITS enable the user to select one or more items, or nothing at all.



Insert Calico Group dialog box

To insert a group:

1. Insert a Configurator Group object into the active document window from the Advisor panel of the Dreamweaver Objects palette.
2. Set the properties of a Configurator Group object in the Insert Calico Group dialog box.
3. Set the following:

ID	Enter an ID that is unique across all controls for your application.
Selection Point	Browse to the <ModelName>.modelinfo.xml file, click it, and choose one of the selection points in the drop-down list box. (The model information file is created when the model is created and compiled.)
Caption Type	Select either <i>text</i> or <i>image</i> from the drop-down list box.
String or Path	If you want the caption to be text, enter it. If you want the caption to be an image, browse to the image, and click it.
Sort	Click to order picked items first in the list after a pick is submitted.
Show Eliminated	Click to display elimination levels, then enter the upper and lower limits, inclusive, for the elimination levels that you want to display. (The modeler defines elimination levels in the model.)
Show “None”	Select one of the items in the drop-down list box. First places the None item at the top of the list. Last places the None item at the bottom of the list. Do Not Show excludes a None item from the list. (This field applies only to single-select lists.)

Replace “None” With

Enter text to identify the item in the list that lets the user choose nothing. “None” is used if you leave this field blank.

Events

Click, then enter the HTML syntax to handle an event for this control. For example:

```
onBlur = "foo()"; onClick= "foo2()"
```

Attributes

Click, then enter the attributes that you want to include. Separate multiple attributes with commas. Attributes replace domain member names. To display attributes, the modeler should create an attribute for domain member names; for example, an attribute called “DomMbrName” or “Desc.”

Note. If you want column headings for your attributes, you must use the Configurator Table object, which creates a radio-box group or check box group, with attributes in columns, and column headings.

Inserting a Table

A Configurator Table is one of the following:

- A radio-button group for a single-select selection point.

This displays the button and attributes for each item in columns with column headings.

- A check box group for a multi-select selection point.

This displays the check box and attributes for each item in columns with column headings.

As with any radio button, the user can select only one item. However, with a Configurator Table object, you can include an item at the top or bottom of the radio-button group to enable the user to select nothing, if the selection point is optional. With check boxes, the user can select one or more items, and can select nothing by clearing all of the check boxes.

<p>DECIDE WHICH VEHICLE</p> <table border="1"> <thead> <tr> <th></th> <th>Code</th> <th>Cost</th> </tr> </thead> <tbody> <tr> <td><input type="radio"/></td> <td>null</td> <td>null</td> </tr> </tbody> </table>		Code	Cost	<input type="radio"/>	null	null	<p>DECIDE WHICH COMPONENTS</p> <table border="1"> <thead> <tr> <th></th> <th>Code</th> <th>Cost</th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/></td> <td>cRTCDS</td> <td>179.99</td> </tr> <tr> <td><input type="checkbox"/></td> <td>cRT</td> <td>99.5</td> </tr> <tr> <td><input type="checkbox"/></td> <td>cRTCDM</td> <td>199.99</td> </tr> <tr> <td><input type="checkbox"/></td> <td>cCDC</td> <td>63.75</td> </tr> </tbody> </table>		Code	Cost	<input type="checkbox"/>	cRTCDS	179.99	<input type="checkbox"/>	cRT	99.5	<input type="checkbox"/>	cRTCDM	199.99	<input type="checkbox"/>	cCDC	63.75	<p>DECIDE WHICH KITS</p> <table border="1"> <thead> <tr> <th></th> <th>Code</th> <th>Cost</th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/></td> <td>kDS</td> <td>12.5</td> </tr> <tr> <td><input type="checkbox"/></td> <td>kT</td> <td>24.25</td> </tr> <tr> <td><input type="checkbox"/></td> <td>kUS</td> <td>17.99</td> </tr> </tbody> </table>		Code	Cost	<input type="checkbox"/>	kDS	12.5	<input type="checkbox"/>	kT	24.25	<input type="checkbox"/>	kUS	17.99									
	Code	Cost																																										
<input type="radio"/>	null	null																																										
<input type="radio"/>	null	null																																										
<input type="radio"/>	null	null																																										
<input type="radio"/>	null	null																																										
	Code	Cost																																										
<input type="checkbox"/>	cRTCDS	179.99																																										
<input type="checkbox"/>	cRT	99.5																																										
<input type="checkbox"/>	cRTCDM	199.99																																										
<input type="checkbox"/>	cCDC	63.75																																										
	Code	Cost																																										
<input type="checkbox"/>	kDS	12.5																																										
<input type="checkbox"/>	kT	24.25																																										
<input type="checkbox"/>	kUS	17.99																																										

A radio button table and two check box tables

DECIDE WHICH VEHICLE is a radio-button table. All of the items in it are null because the modeler did not assign values for the code and cost attributes for this selection point. DECIDE WHICH COMPONENTS and DECIDE WHICH KITS are check box tables. The modeler assigned values to the code and cost attributes for these two selection points. (Code and Cost are actually headings that the web designer assigned to these two attributes. The modeler named these attributes Abbreviation and Price.)

Notice that attributes replace item names. If you want to use the Configurator Table object, the modeler should create an attribute for the name of an item, so that you can display it.

To insert a table:

1. Insert a Table object into the active document window from the Advisor panel of the Dreamweaver Objects palette.
2. Set the properties of a Configurator Table object in the Insert Calico Table dialog box.

ID	Enter an ID that is unique across all controls for your application.
Selection Point	Browse to the <ModelName>.modelinfo.xml file, click it, and choose one of the selection points in the drop-down list box. (The model information file is created when the model is created and compiled.)
Caption Type	Select either <i>text</i> or <i>image</i> from the drop-down list box.
String or Path	If you want the caption to be text, enter it. If you want the caption to be an image, browse to the image, and click it.
Sort	Click to order picked items first in the list after a pick is submitted.
Show Eliminated	Click to display elimination levels, then enter the upper and lower limits, inclusive, for the elimination levels that you want to display. (The modeler defines elimination levels in the model.)
Show “None”	Select one of the items in the drop-down list box. First places the None item at the top of the list. Last places the None item at the bottom of the list. Do Not Show excludes a None item from the list. (This field applies only to single-select lists.)
Replace “None” With	Enter text to identify the item in the list that lets the user choose nothing. “None” is used if you leave this field blank.
Events	Click, then enter the HTML syntax to handle an event for this control. For example: <pre>onBlur = "foo()"; onClick= "foo2()"</pre>
Attributes	Click, then enter the attributes that you want to include. Separate multiple attributes with commas. Attributes replace domain member names. To display attributes, the modeler should create an attribute for domain member names; for example, an attribute called “DomMbrName” or “Desc.”
<hr/>	
	Note. If you want column headings for your attributes, you must use the Configurator Table object, which creates a radio-box group or check box group, with attributes in columns, and column headings.
<hr/>	
Column Heading	Enter the column heading for each attribute that you include. Separate multiple columns with commas.

Inserting an Image

The two image objects enable you to create a control for a single-select selection point that uses images to select a domain member:

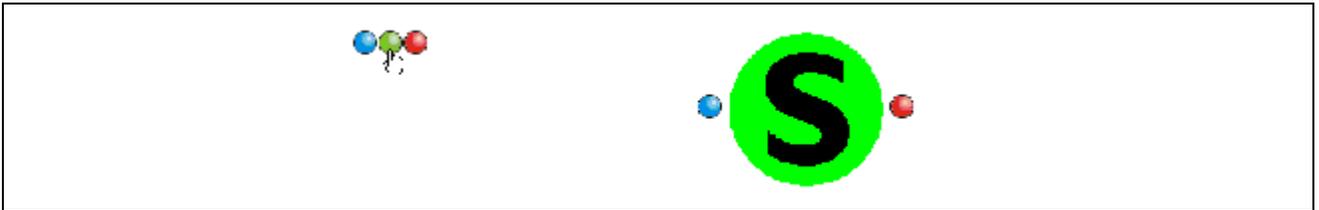
- Single-Select Image Control: Generates an image to select one domain of a selection point.

You use the object multiple times to specify the image and location on the page for each domain member of a selection point. At runtime, selecting one image deselects another.

- **Image Table:** Generates either a column or a row of images for all of the domain members of a selection point. You use it once to specify the image for every domain member of a selection point, and whether you want the images placed horizontally in a row, or vertically in a column. At runtime, selecting one image in the row or column de-selects another.



Single-Select Image controls, before and after selecting an image



Single Select Image Table, before and after selecting an image

When the “selectable” green image is selected and submitted, it is replaced by a corresponding “selected” green image.

Note. The images of a Single-Select Table Image control are placed in a table row (in a table cell) and centered.

You set the properties of a Configurator Image object in one of the following dialog boxes:

Insert Calico Single Select Image Control dialog box

To insert a Single Select Image:

1. Insert an Image object into the active document window from the Advisor panel of the Dreamweaver Objects palette.
2. Specify the properties in the dialog boxes.

Because Image Table generates one control with images for every domain member of the selection point, whereas multiple instances of Single Select Image Control generate a control image by image, the dialog boxes differ from each another in that:

- Single Select Image Control requires a domain member name.
Image Table does not use domain member names.
- Image Table requires an image attribute in the Configurator model.
Single Select Image Control also accepts an image name.
- Image Table requires an alternative text attribute in the model.

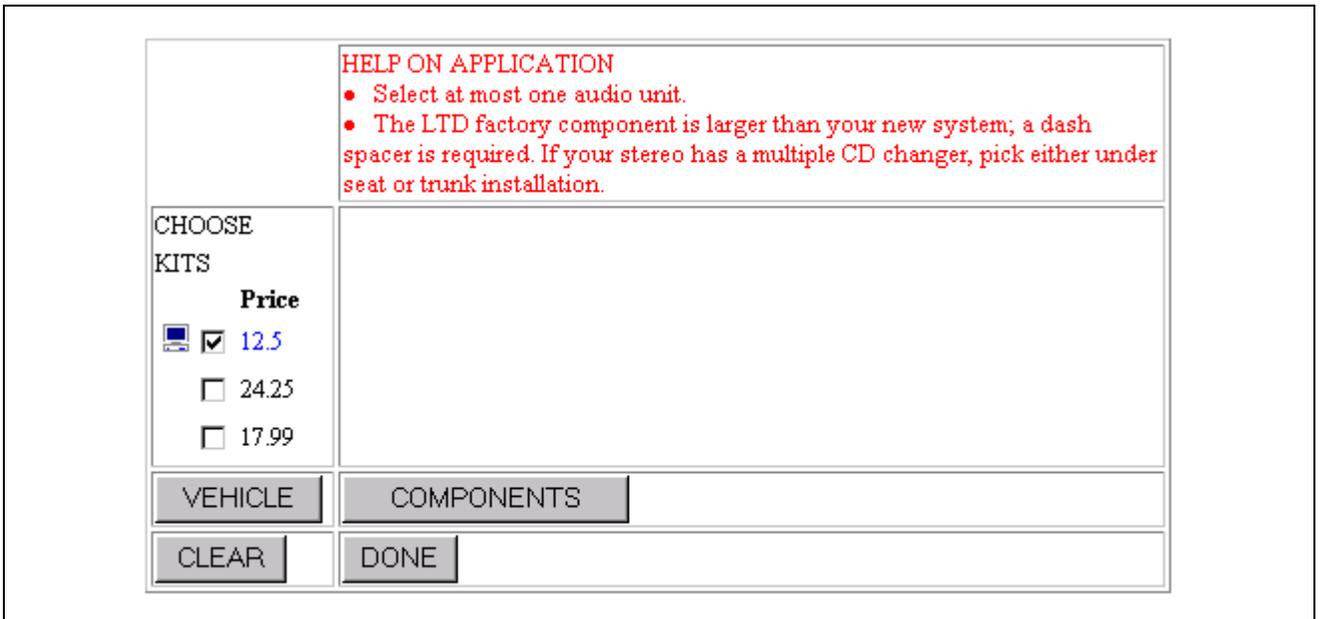
Single Select Image Control also accepts an alternative text name.

Inserting Why Help

Configurator Why Help displays messages on constraints resulting from user picks. There are two types of Configurator Why Help:

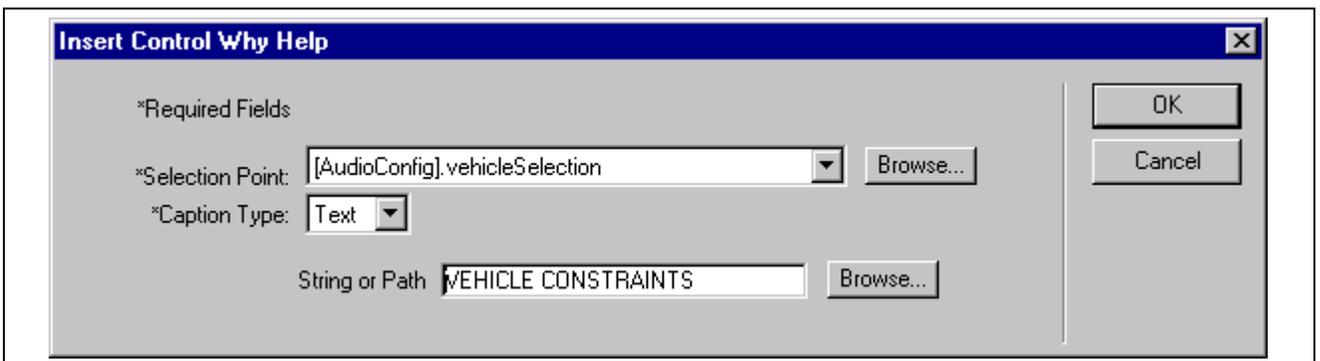
- Control Why Help: This object displays messages on constraints resulting from picks on the specified control.
- Application Why Help: This object displays messages on constraints resulting from picks anywhere in the application.

You can attach labels to the messages. The labels are displayed only when the messages are displayed.

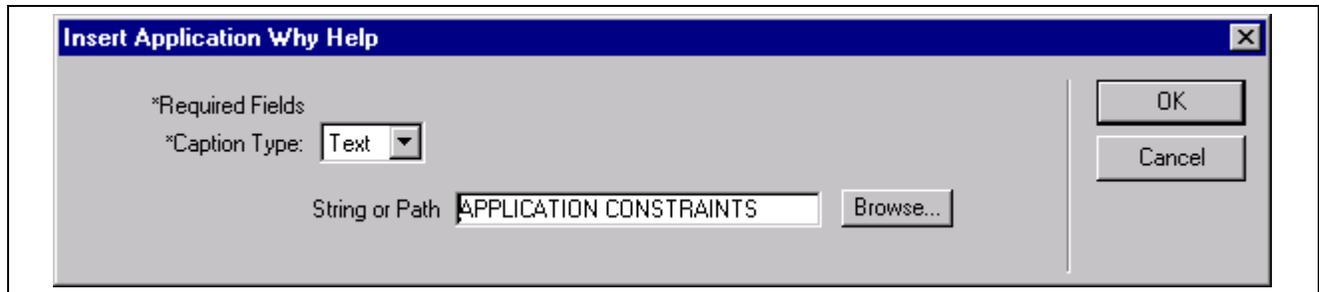


Two Application Why Help messages

You set the properties of a Configurator Why Help object in one of the following dialog boxes:



Insert Control Why Help dialog box



Insert Application Why Help dialog box

Selection Point Browse to the <ModelName>.modelinfo.xml file, click it, and choose one of the selection points in the drop-down list box for which you want Control Why Help. (You do not set a selection point for Application Why Help, because it applies to all selection points in the model.)

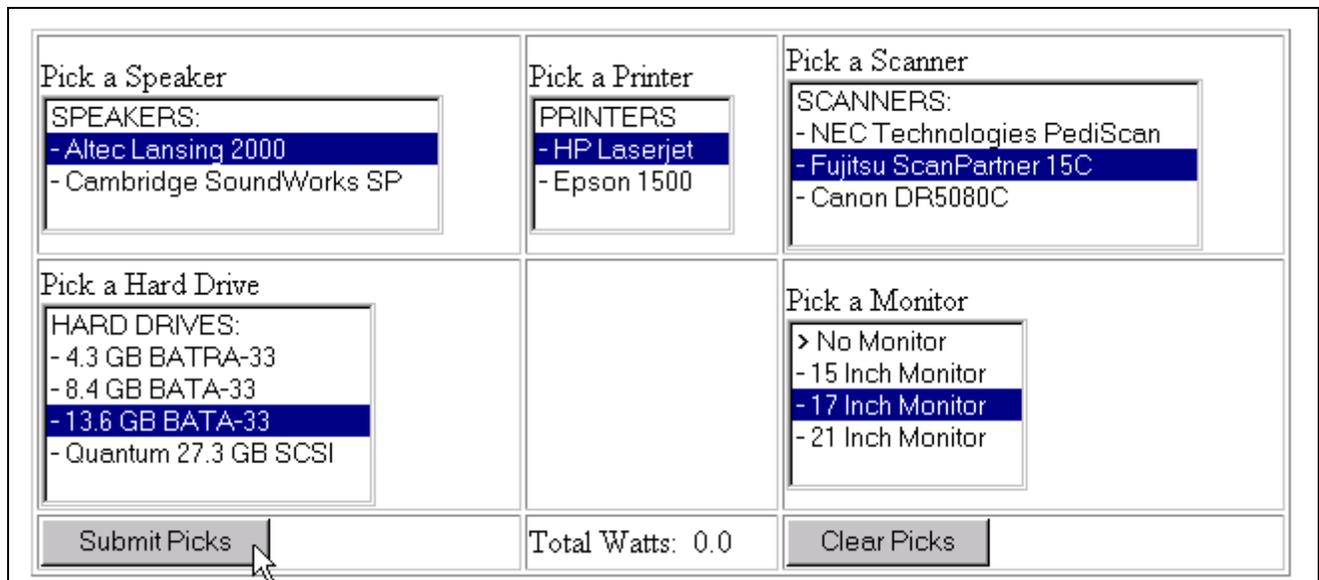
Caption Type Select either *text* or *image* from the drop-down list box.

Inserting a Numeric Data Object

The Configurator Numeric Data object displays a total—that is, a summation—for a numeric attribute of user picks, such as price, at runtime. The summation is preceded on the same line by a caption, followed by a colon (:).

Note. The modeler creates a selection point for each summation.

You can attach labels to the messages. The labels are displayed only when the messages are displayed.



Numeric Data for watts before submitting picks

Pick a Speaker >> Altec Lansing 2000 - Cambridge SoundWorks SP	Pick a Printer >> HP Laserjet - Epson 1500	Pick a Scanner - NEC Technologies PediScan >> Fujitsu ScanPartner 15C - Canon DR5080C
Pick a Hard Drive - 4.3 GB BATRA-33 - 8.4 GB BATA-33 >> 13.6 GB BATA-33 - Quantum 27.3 GB SCSI		Pick a Monitor - No Monitor - 15 Inch Monitor >> 17 Inch Monitor - 21 Inch Monitor
Submit Picks	Total Watts: 13.0	Clear Picks

Numeric Data for watts after submitting picks

Insert a Configurator Numeric Data object into the active document window from the Advisor panel of the Dreamweaver Objects palette.

Set the properties of a Configurator Numeric Data object in the Insert Numeric Data dialog box:

Insert Numeric Data dialog box

Selection Point

Browse to the <ModelName>.modelinfo.xml file, click it, and choose one of the selection points for numeric data—that is, the summation of a numeric attribute—in the drop-down list box. (The model information file is created when the model is created and compiled.)

Note. The modeler creates a selection point for each summation of a numeric attribute, and may constrain the summation to certain selection points. For example, the modeler may constrain a selection point called TotalPrice to the price (a numeric attribute) of HardDrive (one selection point) and PowerSupply (another selection point). At runtime, TotalPrice is the summation of the price attribute of all hard drive items and power supply items selected by the user.

Caption Type

Select either *text* or *image* from the drop-down list box. This caption precedes the numeric data on the same line.

String or Path

If you want the caption to be text, enter it. If you want the caption to be an image, browse to the image, and click it. The Numeric Data caption is displayed only when the numeric data is displayed.

Editing Properties of Advanced Configurator Objects

Each of the Configurator objects, except Form and Button, has a unique Dreamweaver Property inspector that enables you to edit the properties that you set when you inserted the object.



Property inspector for a Configurator List

To access the property inspector for a Configurator object:

1. In an active document window, select the Configurator object whose properties you want to edit.
2. Select Window, Properties.

The Property inspector appears.

3. If the arrow in the lower right-hand corner of the inspector is pointing down, click it to expand the Property inspector.

Editing Forms and Buttons

Configurator forms and buttons don't have a property inspector in which to change their property values. To achieve a form or button with different properties, delete it, then reinsert it, setting the desired properties on the new object.

Warning! The action type for both Configurator buttons is *submit*. Do not set the action type of any Configurator button to *reset* in either the standard Property inspector or HTML source.

Editing Lists, Groups, and Tables

The Property dialog boxes for Configurator List, Configurator Group, and Configurator Table are similar.

CHAPTER 29

Compound Modeling

This chapter provides an overview of the compound model at run time and discusses how to:

- Use the compound model JSP pages.
- Call the compound model API.
- Create an application from the sample.

Understanding the Compound Model at Run Time

PeopleSoft Advanced Configurator offers the functionality for developing a Web application that lets your user configure products based on a compound model.

Creating a compound model involves these steps:

1. Create the standard models.

Using the PeopleSoft Visual Modeler, create a model for each configurable component of your services offering. Decision points in each component can receive values from connected components, and use them in constraints, creating cross-constrained models.

2. Create a compound structure.

Also using the Visual Modeler, define which configurable components from the standard models to include in your offering, and how they connect to one another.

3. Create a user interface.

Using Configurator compound model APIs and JavaServer Pages, create an interface that lets your user dynamically create, configure, and connect instances of your configurable components.

Runtime Capabilities

PeopleSoft Advanced Configurator lets your user not only create a compound model configuration, but change it and generate a change order. It also lets your user save both the original and the revised configuration, and later retrieve information on changes between the two versions.

Configurator JavaServer Pages that you can use in your Web application let your user retrieve information on configuration changes. The pages are described in the following sections:

See [Chapter 29, “Compound Modeling,” Creating an Application from the Sample, page 380](#) and [Chapter 29, “Compound Modeling,” Using Compound Model JSP Pages, page 377](#).

A compound configuration must be able to add, delete, move, and change services (such as Web access or call forwarding in the sample application) on an ongoing basis. To generate a change order, the provider must be able to identify the following changes to the product configuration:

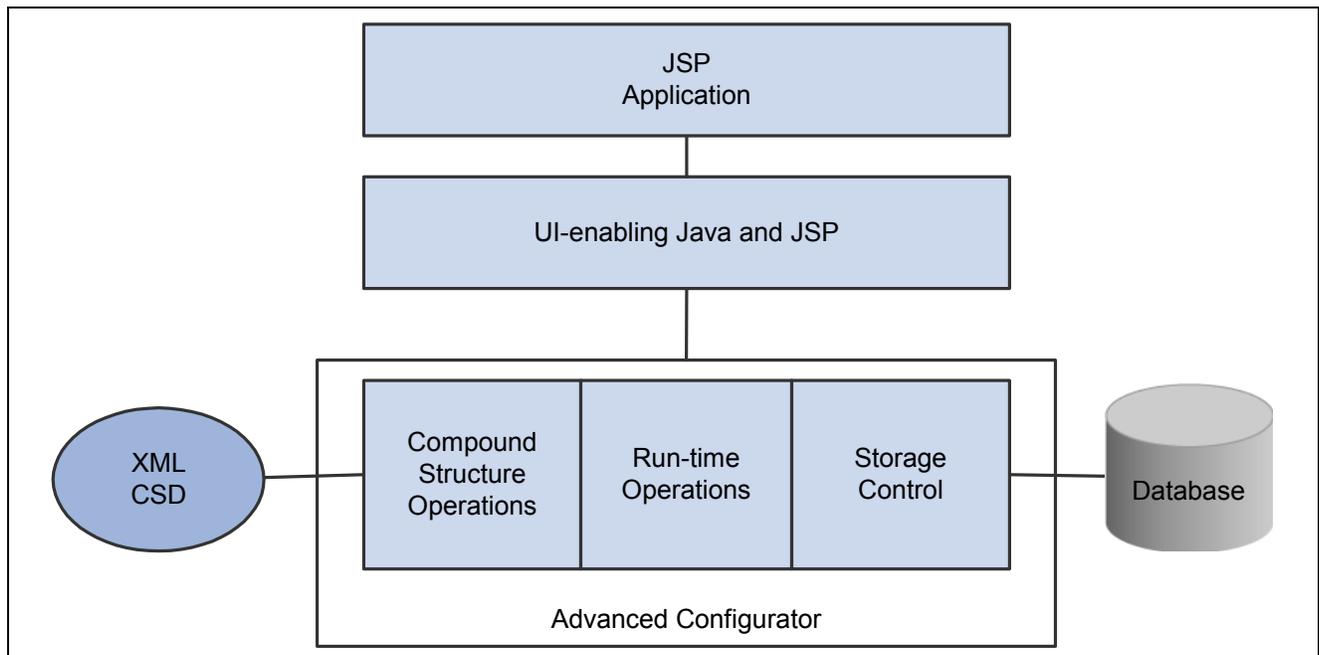
- Whether a component has been deleted.
- Whether a component has been added.
- Whether a component has moved.
- Whether a component has changed and how it has changed.

Deleting, adding, and moving a component is considered a structural change—that is, a change to the structure of the configuration. Revising how a component is configured does not change the structure of the configuration, and is considered to be a component change.

The Configurator has APIs to retrieve information on both structural and component changes made during the user's session. It also has APIs to retrieve configurations by date, which in turn let you retrieve information on changes in the configuration between two points in time.

Architecture

Compound structure components extend the Configurator, which runs on the WebLogic application service. The compound structure extensions include Java classes and JavaServer Pages.



Compound Structure Component Hierarchy

Some Compound Structure Java classes work behind the scenes to let you:

- Create multiple instances of models and configure them.
- Constrain the selection point of one model against the selection point of another model.

Other Compound Structure Java classes give you a public API that lets you create JavaServer Pages that let your user dynamically create, configure, and verify a configuration based on a compound model.

See Also

[Chapter 27, “Using JSP Form Control Templates,” page 325](#)

Using Compound Model JSP Pages

Configurator provides JavaServer Pages that you can include in your Web application to let your user configure components and connect them, and then save the configuration.

In many cases, you may want to change the JavaServer Pages to suit your particular application. They illustrate how to make Configurator calls to accomplish various tasks.

Note. Configurator also has JavaServer Pages for a sample application. The sample application pages use the pages described in this chapter to create a Web application.

See [Chapter 29, “Compound Modeling.” Creating an Application from the Sample, page 380.](#)

If you installed Configurator to the D drive, the JavaServer Pages for configuring and connecting components and for saving a configuration are in the following folder:

```
D:\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\
solutions\Compound⇒
Sample
```

The `networkadvisor` folder includes these categories of JavaServer Pages:

- Components—to let your user add, delete, and change components.
- Connections—to let your user add, delete, and change connections.
- Configurations—to let your user log in, and verify, save, and load a configuration.
- Changes—to let your user view information on changes that have been made to components and connections over time (“delta information”).
- Application—central pages to let your user access the other pages, and pages containing common features.
- Form—to let your user edit the name of a component.

Components

Three JavaServer Pages let your user add, delete, and edit a component. (You can also move a component such as a hub, node, or circuit by editing a connection.)

They are:

```
CalicoNAAddComp.jsp
CalicoNADeleteComp.jsp
CalicoNAEditComp.jsp
```

Connections

Three JavaServer Pages let your user add, delete, and edit a connection:

```
CalicoNAAddConn.jsp
CalicoNADeleteConn.jsp
CalicoNAEditConn.jsp
```

Configurations

Five JavaServer Pages let your user log in, and verify, save, and load a configuration:

Page	Purpose
CalicoNAIdentification.jsp	Lets the user log in to your Web application.
CalicoNAVerify.jsp	Lets the user verify and display errors about whether the configuration is valid.
CalicoNASaveConfig.jsp	Lets the user save a configuration.
CalicoNALoadConfig.jsp	Lets the user load a saved configuration.
CalicoNAErrorInc.jsp	Displays various error messages to the user during the configuration session.

Changes

Two JavaServer Pages let your user view changes made to a compound configuration.

Page	Purpose
CalicoNAVViewDeltas.jsp	Lets the user view which components and connections have been added, deleted, and moved.
CalicoNAVViewDeltaDetail.jsp	Lets the user view a detailed change history for each component.

Application

Seven JavaServer Pages implement the sample application. Because they are included in other JavaServer Pages, their names end with Inc. You can use them as examples for JavaServer Pages to implement your own application.

Two JavaServer Pages help you display and navigate to other JavaServer Pages.

Page	Purpose
CalicoNAMangerInc.jsp	Lets the user view all of the components and connections.
CalicoNANavigateInc.jsp	Displays buttons on the Manager page to let your user choose to log in, and verify, save, load, and view changes to a configuration.

Four JavaServer Pages provide a common set of features for all of the sample application pages, except those used to configure the individual models of the compound configuration.

Page	Purpose
CalicoNAConstantsInc.jsp	A collection of constants used by Configurator JavaServer Pages.
CalicoNACCommonInc.jsp	Contains a collection of common imports and properties for the application object and compound configuration.
CalicoNAHeaderInc.jsp	An include file that you can use to put the Configurator banner at the top of a page.
CalicoNAUtilityInc.jsp	A collection of utility functions, such as those used for logging errors.

Form

CalicoNAFormInc.jsp lets the user edit the name of a component.

Calling the Compound Model API

PeopleSoft Advanced Configurator has an API that lets you do the following:

- Structure a compound model from configurable components.
- Create multiple instances of configurable components.
- Create multiple instances of connections.
- Create, verify, save, and restore a compound configuration.

The Configurator JavaServer Pages contain examples of the use of the API, and can be altered to use in your application.

The Configurator compound modeling API consists of the following packages:

File	Purpose
calico.cms.definition	Handles parsing of the XML compound structure definition (CSD) and provides an API for accessing compound structure information.
calico.cms.dm	The data management package, which is used to access the database in order to store and retrieve compounds, components, and connections, and to retrieve compound delta information.
calico.cms.runtime	Used for managing compound configurations at run time.
calico.cms.exceptions	Contains exceptions thrown by classes in the other compound modeling packages.
calico.cms.cache	Provides methods for accessing and managing the set of compound structures on the server
calico.cms.servlet	Contains a servlet class which provides an HTML interface for managing the compound structures on the server.

The Configurator installation also contains the JavaDoc for the following:

File	Purpose
calico.configurator.cop	This is the Configurator API. It includes modifications that have been made for Configurator.
calico.configurator.exceptions	exceptions thrown by the Configurator API (COP).

Creating an Application from the Sample

The Configurator includes a sample Web application, called TelcoSample, that demonstrates the basic features of a complex product application.

Its JavaServer Pages for the sample are located in:

```
\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\  
solutions\Compound⇒  
Sample\
```

The sample includes:

- XML representation of a compound structure definition that has three configurable components—Node, Hub, and Circuit.
- Source files for the three component models.
- Web application JavaServer Pages that call the compound model API and employ Configurator JavaServer Pages.

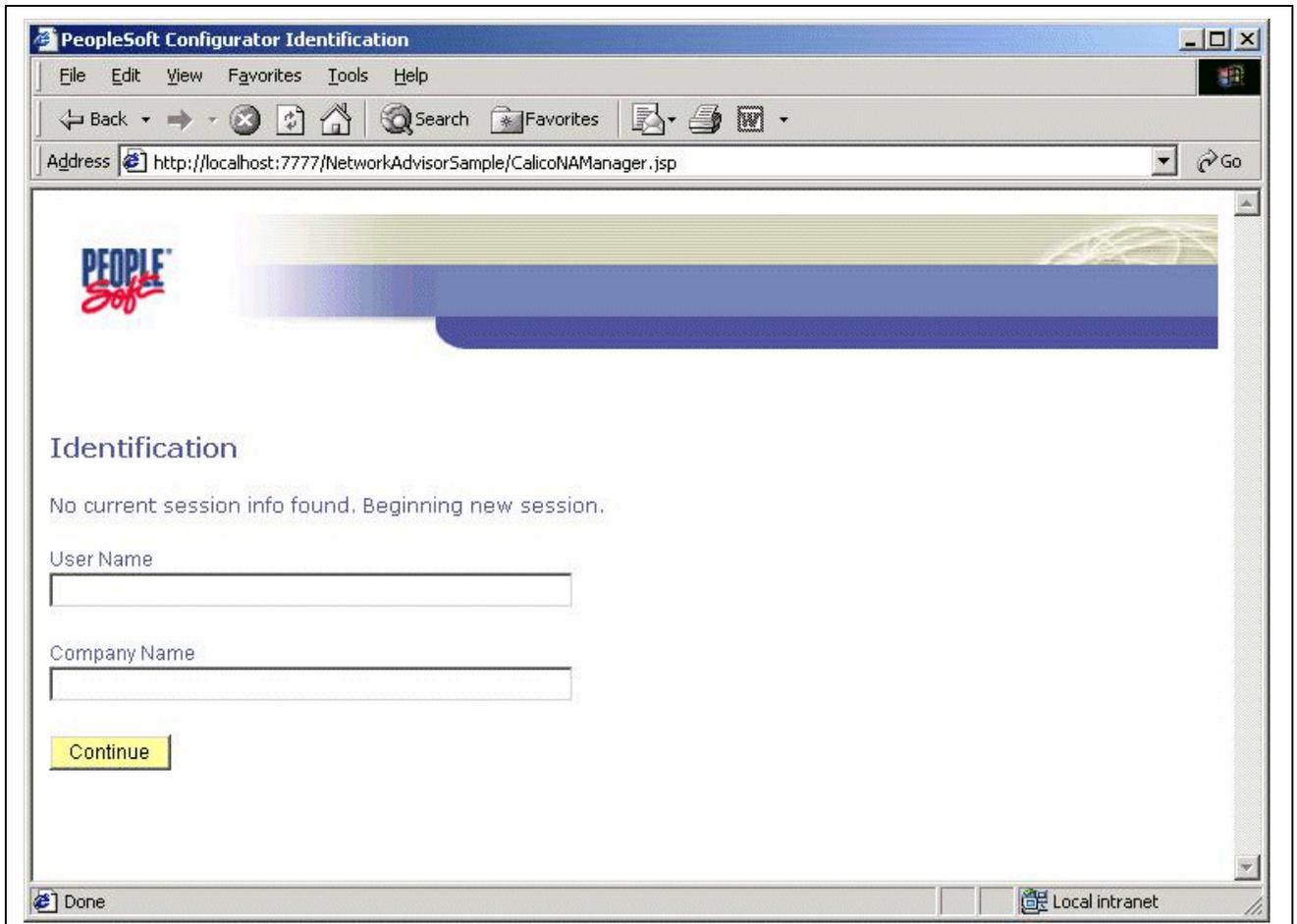
Viewing the Sample Application

This section tells you how to install the sample application, and describes each of its parts. It then shows you how to use the application to configure, save, restore, and reconfigure a network of communication services, and to obtain the delta between the first and second configurations.

To launch the sample application:

1. Make sure that the WebLogic application server is running.
2. From your browser, call the following URL for the application's index page:

```
http:\\<hostname:port>\\solutions\CompoundSample\CalicoNManager.jsp
```



Sample application login page

Node-Hub-Circuit Services

The sample application's compound structure definition has three configurable components, and three connections (or relationships). It does not use Type or Instance elements.

See [Appendix E, "Node-Circuit-Hub Service," XML Representation of Compound Structure Definition, page 468.](#)

Configurable Components

The sample application has three components.

Node

A node is defined by the standard model TelcoComp. By creating a node (using "Add New Component") TelcoComp, and clicking on its name in the Manager page, you can see the decision points and choices of the nodes. The decision points are: Protocol, Access Option, Local Loop Service, CPE, Port Speed, and Local Loop Port Speed.

Hub

A hub is defined by the standard model `TelcoCompHub`. By creating a hub (using “Add New Component”) in your standard model, and clicking on its name in the Manager page, you can see the decision points and choices of the hubs. The decision points are: Protocol, Access Option, and Port Speed.

Circuit

A circuit is defined by the standard model `TelcoCompCircuit`. By creating a circuit (using “Add New Component”) in your standard model, and clicking on its name in the Manager page, you can see the decision points and choices of the nodes. This model defines a circuit with one decision point: circuit speed.

Relationships

The sample application has three relationships.

Connection

A connection is an element of Configurator that can be used to connect to configurable components. You can create new connections (using “Add New Connection”). Each connection is listed on the Manager page, along with the two components it connects. By clicking on the connection name, you can edit it.

The XML CSD (created with the Visual Modeler) for the sample application defines the connection, and in particular what information from one component can be used by another component when they have a connection between them. In this sample application, the CSD says:

- A circuit must have at least 1 and at most 2 connections.
- A circuit can connect to 0 to 2 nodes, and to 0 to 2 hubs (inclusive).

Note. These two rules give us the following possibilities: circuit-node, circuit-hub, node-circuit-node, hub-circuit-hub, and node-circuit-hub.

- A circuit can access the port speed choice of any component it is connected to. The circuit’s names for these port speed choices are `Ext_PortSpeedSelection_A` and (if the circuit is connected to a second component) `Ext_PortSpeedSelection_B`.
- A circuit can access the protocol choice of any component it is connected to. The circuit’s names for these protocol choices are `Ext_ProtocolSelection_A` and (if the circuit is connected to a second component) `Ext_ProtocolSelection_B`.

In its own (standard) model, the circuit can use these port speeds and protocols to constrain its own domain members. This is how cross-constrained models are created.

HubCircuits

According to the XML CSD definition of the `HubCircuits` relationship for the sample application, a hub can be connected to 1 to 24 circuits (inclusive). The hub has access to the collection (set) of the port speeds of all the circuits connecting to it. The hub’s name for this set is `Ext_PortSpeeds`.

In its own (standard) model, the hub can use `Ext_PortSpeeds` to constrain its own domain members. Because `Ext_PortSpeeds` is a set, the model will probably apply an operator like “max” or “sum” to it.

NodeCircuit

According to the XML CSD definition of the `NodeCircuit` relationship, each node must be connected to exactly one circuit.

Modeling Node-Hub-Circuit Services

The sample application has three models with cross-constraints:

- TelcoComp—for a node.
- TelcoCompHub—for a hub.
- TelcoCompCircuit—for a circuit.

You can look at each model and see the use it makes of the cross-constraints that compound modeling makes available. For example, TelcoCompHub has a constraint that says “the port speed of the hub must be greater than or equal to the maximum of the port speeds of all the circuits connecting to the hub.”

Configuring Node-Hub-Circuit Services

The sample application lets the user do the following:

- Create a compound Node-Hub-Circuit configuration, and save it.
- Restore a saved configuration, and reconfigure it.
- Obtain the delta between the first and second configuration.

Creating a Compound Configuration

On the Load Configuration page, use the New Configuration button to create a new configuration. This displays the Manager page. On the Manager page, create the components (in the sample application, hubs, nodes, and circuits) by using the Add New Component button. Then create connections between these components by using the Add New Connection button.

Reconfiguring a Compound Configuration

To add a new component or connection, use the Add New Component or Add New Connection button on the Manager page.

To delete a component or connection, click on that component’s or connection’s trash can icon on the Manager page.

By clicking on the name of a connection on the Manager page, you can edit the connection, including changing its name and changing which components it connects. You can move components in your compound model by editing the connection that connects the components.

By clicking on the name of a component on the Manager page, you can edit the component, including changing its name and making new choices for its decision points.

Obtaining the Configuration Delta

Clicking on the “View Deltas” button on the Manager page will take you to the View Deltas page, which has a list of all the model’s components and connections and their modification date. By clicking on the name of a component or connection, you can view more detailed delta information for that element.

PART 7

Mobile Product Configuration

Chapter 30
Understanding Mobile Product Configuration

Chapter 31
Mobile Solution Administration

Chapter 32
Creating and Maintaining Mobile Solutions

CHAPTER 30

Understanding Mobile Product Configuration

This chapter provides an overview of mobile product configuration and discusses:

- Mobile configuration.
- Solution package contents.
- Functional components.
- Data.
- Solution distribution and installation.
- Types of models.
- Mobile Configurator Packaging Tool.
- Mobile machine requirements.
- Package maintenance.

Mobile Configuration

PeopleSoft Enterprise Mobile Configurator makes offline product configuration possible for field sales personnel who must configure products without benefit of a runtime connection to the enterprise server.

Mobile components of Advanced Configurator allow a mobile user to download complete and up-to-date product and model information for later local machine configuration. When the user reconnects with the enterprise, he or she can upload the new configurations and transaction artifacts to the enterprise for storage or further processing.

Mobile Configurator components support the creation and maintenance of a browser-based application that solves an business problem requiring off-line configuration.

In this document, the collection of application pages, models, and supporting files that solve an business problem, such as product configuration or product recommendation, is referred to as a *Solution*. For file management by the Packaging Tool, the Solution is defined in a *Solution Package* file having the extension *.cci*. The Package contains a list of files and their storage locations.

The end product of the Packaging Tool is an installer of component files called the *Solution Installer*. Generated from the Solution Package, the Solution Installer is an *.msi* (Microsoft installer database) file containing all of the files and information that the mobile user needs to install and run the Solution on a local system installed with PeopleSoft Mobile Configurator. The *.msi* is the file that is distributed to the field, either by email attachment, media, or download.

Note. Model requirement: To accommodate a local, offline, runtime environment, models intended for mobile deployment must have data internal to the model. The Configurator Administration console provides a function to enable the modeler to quickly internalize model data.

See [Chapter 31, “Mobile Solution Administration,” Internalizing Model Data, page 396](#).

Solution Package Contents

From the Solution Package (.cci), the Packaging Tool creates a Microsoft installer database file (.msi) containing all of the components needed to run a Configurator application on the Mobile client:

- Model files (runtime model).
- Pages (<modelpage1...n>.jsp).
- Images.
- Directory information for the above Solution files.
- Installation information for all of the above.

Model files, pages, and images are compressed into the .msi install file. Because the .msi file contains an install script, double-clicking on the file executes a Windows 2000 installation program that provides the mobile user with a ready-to-run application for configuration.

Functional Components

Mobile Configurator components are:

- Mobile Configurator Packaging Tool—Allows a modeler or model administrator to create an install file containing all of the components required to carry on a customized PeopleSoft configuration application session on Mobile Configurator.
- Mobile Configurator client—The client application from which the mobile user configures the product. Online documentation accessible from the application describes its purpose and how to use it.
- Transfer Service—Allows transfer of configuration data from the Configurator Engine to other third-party ERP systems for further data processing. The Transfer Service API includes sample implementations to aid in customizing data exchange.

Data

The Mobile Configurator is designed to manage the various bodies of data with minimal intervention. For example, the Packaging Tool acts as a central management point for model, site, and related support files. The Mobile client offers mobile users straightforward upload and download of configurations. And for modelers, an internalizing function eliminates the need for database-referenced model data.

Model Data

Data specific to configuring the product is in the form of:

- Domain member and constraint data.
- Domain-member attribute data for display and downstream use that is not handled by the Configurator engine but is needed for configuration.

Accessing this data for configuring in a mobile environment requires that the data be local—specifically, it must be internal to the model. You can internalize data in one of two ways:

- Assign domain member values manually in the Visual Modeler, OR
- Use the model internalization function of the Advanced Configurator Administration Tool to automatically populate domain member values from an external database, such as the enterprise central product store. Then re-compile the model and update the Solution document with the new model version.

The internalization function lets you transfer the latest body of product data from the enterprise database to the model file itself.

See [Chapter 31, “Mobile Solution Administration,” Internalizing Model Data, page 396](#).

Site Support Data

Non-configuration data that is referenced in the JSPs is included in the Solution by the site developer rather than the modeler or model administrator.

Configuration Data

Configuration data is the result of a configuration session. The mobile user saves the configuration to local storage for later upload to the enterprise. The Mobile client also gives the user the option of emailing the configuration.

Once uploaded to the enterprise, the configuration data can be routed for storage management, quoting, order entry, or other processing.

Solution Distribution and Installation

There are many ways to distribute the Solution Installer (.msi). For instance, you can:

- Post the .msi file for online distribution by FTP or HTTP protocol.
- Distribute it as an email attachment.
- Transfer it using removable media.

The .msi file is an installation executable; double-clicking the .msi file installs the Solution on the mobile machine.

Note. A Solution Package will not install on a machine unless the Mobile Configurator has been installed.

Types of Models

At the heart of any PeopleSoft Configurator Solution is the model. A Solution may contain one or more component models, which are programmatic definitions of a product and its options, the relationships that describe and limit the combination of those options, and associated business data such as price and availability.

Often, business data, being changeable, is provided to the model at run time (during a configuration session) by an outside source such as an enterprise database. This is external data and delivers the most current information to the user. Sometimes a model's data is stored within the model itself as internal data.

If you are packaging a Solution, you need to know if the model has external data that needs to be internalized before packaging or updating it.

See [Chapter 31, “Mobile Solution Administration,” Internalizing Model Data, page 396](#).

If a product is particularly complex, a Solution may contain one or more compound models. A compound model describes the relationships between two or more component models that together describe the product. The description is expressed in XML; the compound model is the XML file resulting from its compilation by the Visual Modeler.

If a compound model(s) is the type of model employed in a Solution, then the subdirectory containing its .xml file is the root directory for the entire Solution, even if other parts of the Solution, including its component models, are located elsewhere. This is true because the compound model XML file contains the locations of its component models.

Mobile Configurator Packaging Tool

The Mobile Configurator Packaging Tool allows you to:

- Create and maintain a Solution Package—a list of the files that comprise a Solution.
- Create a Solution Installer for the purpose of distributing Solution Package files to a mobile sales force.

Creation and maintenance of a Solution Package entails managing the name, version, and location of the desired files, as well as adding and deleting files. While relative path information is automated for core Solution files, you have the flexibility to define and manage the relative organization of miscellaneous files within a Package.

The tool gives you these basic capabilities:

- Create new Solution Packages.
- Modify existing Solution Packages.
- Generate Solution and model-update installers.

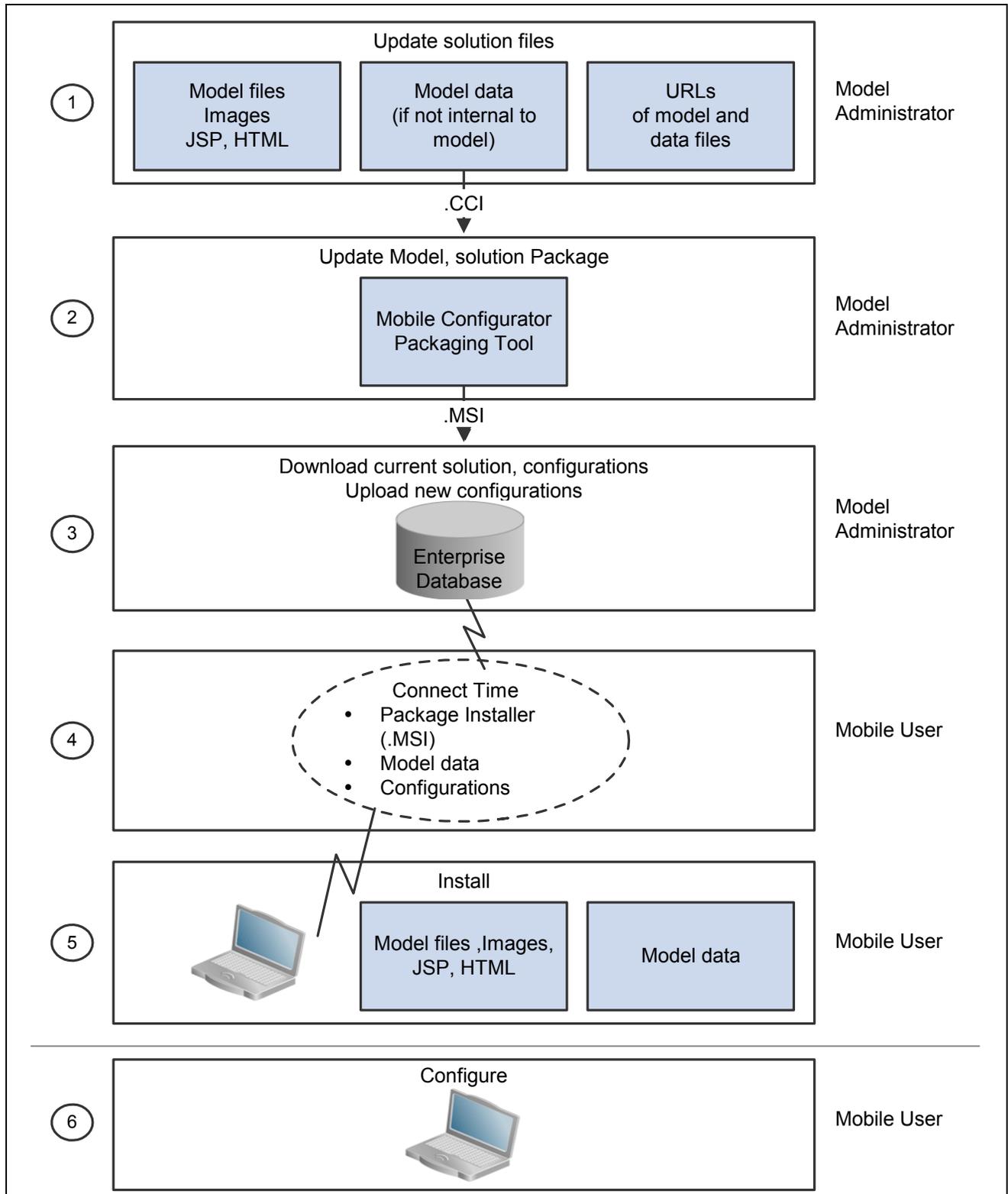
Inputs required by the Mobile Packaging Tool for packaging a Solution are:

- Model files and the image and page files of the Solution UI.
- Locations of the aforementioned files for purposes of copying and compressing them into an install package.

The tool's output is a .cci file (a description of the Solution contents) and an .msi file (the Solution Installer).

Workflow is as follows:

1. A new Solution is created, or files in an existing Solution change, requiring an update to field installations. Changes can involve the model itself (such as constraints and relationships), model data (domain members and pricing), or the presentation pages (JSP and images). Any external data is internalized to the model.
2. The model administrator uses the Packaging Tool to describe the Solution files (.cci) and create a Solution Installer (.msi) from the description, thus replacing the previous installer with one containing the most current files.
3. The Solution Installer is transferred to distribution media, a designated distribution location on the enterprise server, or is simply attached to an email.
4. The mobile user receives the new installer by email, media, or download from the enterprise server, and installs it. While connected to the server, the mobile user can upload configurations to the enterprise server.
5. The mobile user double-clicks on the .msi file and the Solution application is automatically installed on the local machine.
6. Now disconnected from the enterprise server, the mobile user launches the updated Solution application.



Mobile configuration data flow

Mobile Machine Requirements

In order to run the Configurator Solution installed by its associated Solution Installer, the mobile machine must meet the following requirements:

- The operating system is Windows 2000 or Windows XP.
- Mobile Configurator must be installed.

PeopleSoft provides a separate installer for Mobile Configurator.

See *PeopleSoft Enterprise CRM 9 Installation Guide*

Package Maintenance

Solution update and maintenance requires more thoughtful care when maintaining models hosted to numerous off-site machines beyond your control and oversight.

Controlling Field Version Use

The site or model developer can specify options in the Configurator to allow mobile users the flexibility of choosing which Solution version to use to create or edit a configuration. Such a choice allows the mobile user to modify existing configurations that have been superseded by product updates. In addition, simply opening an existing configuration for verification requires that it be restored in the UI it was created with.

See [Chapter 27, “Using JSP Form Control Templates,” Specifying Solution Information Properties, page 349.](#)

Designating Solution versions is a matter of specifying the version policy settings in `CalicoUI.properties` (for component models) and `CalicoNA.properties` (for compound models). You should be aware that the Mobile client gives the mobile user the choice of whether or not to use the compile version of the requested configuration.

Guidelines for Maintenance

The Packaging Tool allows you substantial flexibility in creating and updating installers that track changes throughout the life of a given Solution. To avoid version confusion and file-sharing problems, it is important to follow these guidelines:

Don't put the same files in multiple packages.

Any file that is shared between two packages has the potential for causing installation dependencies. Because the installers created by two such Packages will overlap on the shared file. Thus, a machine with both packages installed will fail to properly un-install either package. To avoid this problem, keep all shared files in one package. You can then upgrade this package as needed and send the upgrades to the field in conjunction with other packages.

Avoid putting more than one Solution in a package.

This will greatly reduce the risk of Solutions overlapping other packages.

Keep and maintain your Packages for the life of a Solution.

When packaging a Solution for the first time, keep the Package for later updates. Once a new version of the Solution is available, update the original Package file with the changes to the Solution. In this way, the new installer will update previous installations on the mobile machine.

The Solution Installer will upgrade the previous version because it contains identification codes unique to each Package and installer. The Packaging Tool manages these codes for you. Every time you save changes to a Package, the codes are updated. These codes enable the Microsoft installer application to distinguish between alternate installations of the same Solution or upgrade previous installations of the Solution.

When you create a Model Update Installer, be aware that you are creating a new product installation.

The menu command Tools, Create Model Update Installer creates a new Model Update Installer, which will appear as a separate installer (.msi) on the mobile user's machine. By contrast, generating an installer with Tools, Create Solution Installer generates an installer that simply upgrades any previous version of that Solution on the mobile machine.

The Packaging Tool uses two templates, both of which are .msi files:

- blank.msi
- confirm.msi

CHAPTER 31

Mobile Solution Administration

This chapter discusses how to:

- Prepare a solution for packaging.
- Localize solution UI files.
- Customize and localize the Mobile Configurator Client.
- Integrate Mobile Configurator with ERP systems.
- Integrate Mobile Configurator with Mobile Order Capture.

Preparing a Solution for Packaging

Before a Solution can be packaged, it must be prepared for distribution and operation in a mobile environment, and for the transfer of configuration data between mobile and enterprise data stores.

Very little is required to adapt an existing PeopleSoft Configurator Solution for deployment in a Mobile Configurator environment, provided the Solution uses JavaServer Pages.

Depending on the Solution's implementation, you may need to do one or more of the following tasks:

- Model data must be made specified in the model itself, and not requested from a database or another file. Such externally stored data can be internalized using the Configurator Administration console or Visual Modeler.

See [Chapter 31, "Mobile Solution Administration," Internalizing Model Data, page 396](#).

- The restore policy and start page must be set in the CalicoUI.properties (for component models) or CalicoNA.properties (for compound models). You can also set other, optional solution parameters in these properties files. These parameters are:
 - Name (the solution ID).
 - Description.
 - Restore page.
 - The directory that the Solution resides in.
 - Language and country codes.
 - Enable switch to allow the Solution to create new configurations.
 - Solution's UI Version.
 - Compound model name and version
- If the intended user is non-English-speaking, the Mobile Configurator client must be localized.

See [Chapter 31, “Mobile Solution Administration,” Localizing Solution UI Files, page 398](#).

- If configuration data is to be passed to and from the components of an ERP system, the Mobile Transfer Service must be integrated.

See [Chapter 31, “Mobile Solution Administration,” Setting Configuration Restore Policy, page 397](#) and [Chapter 31, “Mobile Solution Administration,” Integrating Mobile Configurator with ERP Systems, page 400](#).

You do *not* need to make additions to the UI files. You also do not need to re-stage the files, as the Packaging Tool can extract them from their deployed or staged locations, however they are distributed.

See Also

[Part 2, “Product Modeling with a Component Model,” page 9](#)

Internalizing Model Data

The Advanced Configurator Administration console can internalize model data for you. It can import externally referenced model data and translate it into internally defined data. The model then becomes self-contained, and can be deployed in a mobile configuration environment needing no access to the product database.

You can also include additional attributes in the internalization process through the use of an additional attribute file. Use this text file to specify data that the user interface requires but is not in the model itself. Typically, external attribute data is queried from the JSP page rather than provided in the model.

The additional attribute file lists the additional classes to be internalized using the format:

```
<className>\:<attributeName>=<type>
```

where *type* is *String*, *int*, *float*, or *Boolean*.

An example of an additional attribute file is:

```
Tires\:description=string
Tires\:sidewall=boolean
Tires\:mfg=string
Wheels\:Description=string
Wheels\:mfg=string
```

Note. The Visual Modeler can also internalize data; however, it can only internalize data used within the model. Externally referenced data not used in the model, such as descriptions, can't be internalized by the Visual Modeler.

To internalize model data:

1. Start the Administration console by selecting Start, Programs, PeopleSoft Applications, Configurator, Administration. Make sure the Configurator Server is started.
2. Select the check box for “Generate self-contained model” to internalize model data.
3. If there is a separate file containing additional external attribute information, enter it in the Additional attribute file field.
4. Click Compile. The Administration console internalizes data referenced by the appropriate classes in the model as well as any specified in an external attribute file.

Setting Configuration Restore Policy

A user's request to restore a saved configuration presents a challenge in light of the requirement for that configuration to run on the same, or a compatible version of, the model that it was created with. This is necessary in order to avoid a situation in which picks on the saved configuration do not appear because their corresponding selection points were removed in more recent model updates.

To give you control over which model version is to be used when a saved configuration is restored, a set of standard policies has been established.

The policies are keyed by number as follows:

Policy	Model Type	Action	If not available...
1	Component, Compound	Try to use the same version that the configuration was created with.	Fail
2	Component, Compound	Try to use the same version that the configuration was created with.	Use the newest version.
3	Component, Compound	Always use the newest version.	Fail
4	Component	If more than one version exists, return status, if one uses that version.	Fail
	Compound	Try to use the same version that the configuration was created with.	Automatically use the version in the structure document.
5	Component	Allow the use of any model through specific action (Open with...), or setting.	Fail
	Compound	Always use the version in the structure document.	Fail

To implement one of these policies, edit `CalicoUI.properties` (for component models), or `CalicoNA.properties` (for compound models), as follows:

- Assign a policy number (1 through 5) to `calico.solution.restorePolicy`.
- Assign a value to `calico.page.restore`.

Note. The `CalicoUI.properties` and `CalicoNA.properties` files for an application are located in their respective application root directories.

See Also

[Part 2, "Product Modeling with a Component Model," page 9](#)

[Part 6, "Building a Custom User Interface," page 295](#)

Localizing Solution UI Files

PeopleSoft Configurator provides templates for localizing text in the Solution UI files. In addition, handling the installer (.msi) files conforms with Win2K methods and practices.

Text strings for localizing the Solution Installer are found in:

- blank.msi—for Packages created with no UI.
- confirm.msi—for Packages created with a configuration UI.

Make sure that these files are in the same directory as the SolPkg.exe. For more information on localizing your implementation, please refer to the section entitled “Localizing a Windows Installer Package” of the Platform SDK Windows Installer documentation in the MSDN library.

Customizing and Localizing the Mobile Configurator Client

There are a number of reasons you may want to customize the content displayed by the Mobile client. You may want the Mobile client to display your company logo, sales info, or other custom content when it is idle. You may want a way for your mobile user to quickly log into your company support site. You may have a download page that allows your mobile user to quickly access new or updated Solution Installers and other content. This is done easily using standard web page authoring tools and techniques.

The PeopleSoft Mobile Configurator client can be localized and customized without the need to obtain a new version of the executable from PeopleSoft.

Using external HTML files, you can customize these Mobile client UI elements:

- Main window background.
- The target of the Help menu’s On the web item.
- The target of the Help menu’s Updates item.

Customizing the Mobile Client

The Mobile client uses HTML content for the client background area displayed when the client is idle (not actively configuring a product or displaying other content) and when the New Configuration dialog is being displayed.

By default, the Mobile client reads the HTML content from data contained within the client executable (cmc.exe). However, when the Mobile client is launched, it checks for external HTML files in the installation root directory (default location C:\Program Files\PeopleSoft Applications\Mobile Configurator). If it finds external HTML files for this content, it will use them rather than the internal data.

The content for the Help menu’s On the web and Updates items can be replaced using external HTML files. The Updates menu item is only displayed if an external HTML file exists.

To customize the content displayed by the Mobile Client:

1. Use standard web page authoring tools and techniques to create the content page.
2. Name the content page file according to its display requirements, as follows:

Content shown when...	External file name
Mobile client is idle.	Idle.htm
Mobile client is displaying the New Configuration dialog.	FileNew.htm
User accesses the On the web Help menu item	OnTheWeb.htm
User accesses the Updates Help menu item (the Updates menu item is only displayed if this external file exists)	Updates.htm

3. Place the content page file in the Mobile client installation root directory. The default installation root directory is C:\Program Files\PeopleSoft Applications\Mobile Configurator (where the cmc.exe) is located. Be sure to include any additional files that the content page is referencing, such as image or sound files.

Note. If you intend to build an installer for these files, you can locate the Mobile client installation root directory path in the Windows Registry on the client machine under the key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\PeopleSoft\PeopleSoft Applications
\CfgMobile\InstallRoot
```

The Mobile Configurator client uses the Microsoft Internet Explorer web browser to display the content in the client area and the pop-up windows, so any content appropriate for IE 5.5 (or newer, depending on what is installed on the client machine) can be displayed by the Mobile client. However, keep in mind that the Idle.htm and FileNew.htm content will probably be displayed while the client machine is disconnected and does not have access to the network or Internet, so you should take care to create pages that are displayed correctly when access is unavailable.

Tip for setting up OnTheWeb.htm and Updates.htm

The page the Mobile client initially displays is from the local file, which limits what you can display. To extend display possibilities, include a redirect on the local page to online content or provide links to online content as appropriate.

Localizing the Mobile Client

By default, the Mobile client menu text, dialog text, and messages are in English. Using a Resource Only DLL, you can localize the Mobile client to present text for another language.

The Mobile client menu text, dialog text, messages, and other content are accessed from a Microsoft format resource package. The default resource package is contained within the executable (cmc.exe). However, the Mobile client can load the resources from an external resource package in the form of a Resource Only Dynamic Link Library (DLL). When the Mobile client is launched, it checks for an external DLL file, cmc.dll, in the installation root directory (by default, C:\Program Files\PeopleSoft Applications\Mobile Configurator). If it finds cmc.dll, it loads the resources from the external resource package rather than from the internal package.

For proper operation, all of the Mobile client resources must be present in the external package.

Any tool that can create a Windows resource package in the form of a DLL can be used to create the cmc.dll file. The Mobile Configurator administration installation includes a Microsoft Visual Studio Resource Only DLL project for you to use as a starting point for a localized version.

To localize the client:

The following procedure assumes that you are familiar with how to edit and build a Resource Only DLL.

Warning! When making changes to the `cmc.rc` as described below, be sure that you don't change any resource Ids.

1. Create a copy of the sample project and open it with Microsoft Visual Studio (Visual C++). If you are using a tool other than Microsoft Visual C++ create an appropriate project using the source files included with the sample.
2. Open the `cmc.rc` file:
 - If your localization involves only changing text strings, you can open the `cmc.rc` file in text mode and modify the strings directly using the text editor. In addition, you may want to change the default HTML content used by the Mobile client as part of your localization.
See [Chapter 31, "Mobile Solution Administration," Customizing the Mobile Client, page 398.](#)
 - If your localization requires changes to the size or layout of the dialogs, you can open the `cmc.rc` file using the resource editor rather than the text editor.
3. Replace the English text with text appropriate for your locale.
4. Modify or replace icons and other graphics as needed for your localization.
5. Build the Resource Only DLL. (The details of how to build the DLL depend on the tool used and are beyond the scope of this document.)
6. Copy the resulting `cmc.dll` file to the directory containing the Mobile client executable (`cmc.exe`). Typically, this is the Mobile client installation root directory. By default, the installation root directory is `C:\Program Files\PeopleSoft Applications\Mobile Configurator`.

Note. If you intend to build an installer for these files, you can locate the Mobile client installation root directory path in the Windows Registry on the client machine under the key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\PeopleSoft\PeopleSoft Applications\CfgMobile
\InstallRoot
```

Integrating Mobile Configurator with ERP Systems

Mobile Configurator can interface to the Mobile Transfer Service (MTS), which runs on the enterprise Configurator Server. You can configure an instance of the MTS to provide facilities for transporting configurations and configuration-derived artifacts from Mobile Configurator sites to the enterprise for storage management, quoting, order entry, or other processing.

Through an HTTP POST, the Mobile client passes configuration data to the MTS as XML in the `.cmc` file. The request invokes the Receiver servlet that is running inside the MTS. The servlet in turn invokes the Reader, Handler, and Response implementations specified in the `MobileTS.properties` file to process the request.

The MTS provides an API for creating site-specific implementations.

MTS Interfaces

The Mobile Transfer Service APIs are:

- `calico.mobilets.MobileTSResponse.java`

Interface for a Response. It provides status and content from the uploaded file.

- `calico.mobilets.MobileTSReceiver.java`

Interface for a simple rfc1867 upload receiver. It reads an HTTP POST request having a content type of multipart/ form data. It composes an HTTP response containing the HTTPRequest and MIME, content headers, and any additional data about the request.

- calico.mobilets.MobileTSReader.java

Interface for a MimeReader.

- calico.mobilets.MobileTSHandler.java

Interface for a Handler. Site-specific implementations process the uploaded configuration files.

Sample Implementations

Also provided are samples for reference as you create your own implementations.

The sample implementations are located on the PeopleSoft Advanced Configurator install in `\samples\Configurator\MobileTransferService\Release\calico\mobilets`.

Implementations are:

- MobileTSResponseImpl.java

A sample MobileTSResponse implementation. Returned by MobileTSHandler.handle().

- MobileTSMultipartReaderImpl.java

Reads a stream of MIME multipart/ form data.

- MobileTSHandlerImpl.java

Stores the uploaded file to the local file system in the directory specified in MobileTS.properties.

To use the Transfer Service with these implementations, you must place the implementation class files *either*:

- In a location that is in the WebLogic server's classpath, *or*
- In any location and add that location to the WebLogic server's classpath.

Using the Transfer Service

To hook up the Transfer Service to pass configuration data to ERP receivers, first implement the MTS interfaces as required by your ERP system and Solution.

In MobileTS.properties, edit the values for the Handler and Reader classes to reference your implementation classes.

Default location for MobileTS.properties is: `C:\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\Web-inf\config`

Specify the location of the configuration data file when it is uploaded.

The text of the MobileTS.properties file is:

```
#####
# PROPERTIES FOR CALICO MOBILE - TRANSFER SERVICE
#####
# Handler Class
HandlerClass=calico.mobilets.MobileTSHandlerImpl
# Reader Class
ReaderClass=calico.mobilets.MobileTSMultipartReaderImpl
```

```
# For sample Handler.  
# Directory on local file system where the Transfer Service Sample  
Handler places the uploaded file.  
# Note: Please use forward slash for separator.  
UploadDirName=C:/Temp
```

Using Source Control

If your Solution files are stored in source control, be sure to include Solution Package (.cci) files as well. It is not necessary to protect the Solution Installer (.msi) files in source control for two reasons: they are not editable, and they can be regenerated from the .cci file.

Integrating Mobile Configurator with Microsoft COM-Compatible Applications

Mobile Configurator includes a COM API, which allows another application to interact with the Mobile Configurator to create, save, and restore configurations.

A description of the API's class and methods is available.

See Also

[Appendix F, "Mobile Configurator COM API," page 471](#)

CHAPTER 32

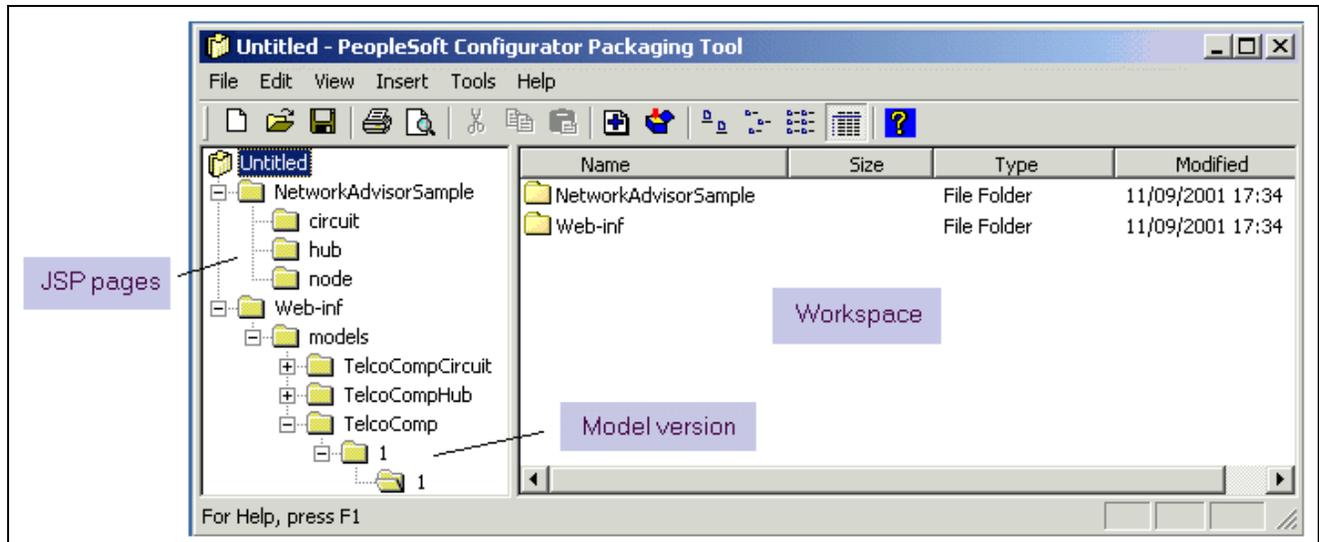
Creating and Maintaining Mobile Solutions

This chapter provides an overview of the Packaging Tool and describes how to:

- Set the application root.
- Create a new solution package.
- Add a model to the package.
- Set end-user installation options.
- Modify a solution package.
- Add non-solution files to the solution package.
- Access package properties.
- Add annotation to the package.
- Generate the solution package installer.
- Update a solution installer.
- Update only the model(s) of a package.
- Print solution package content.

Understanding the Packaging Tool

This figure shows the Packaging Tool's main window, which is a workspace for managing the Solution files for the creation and maintenance of a Solution Package.



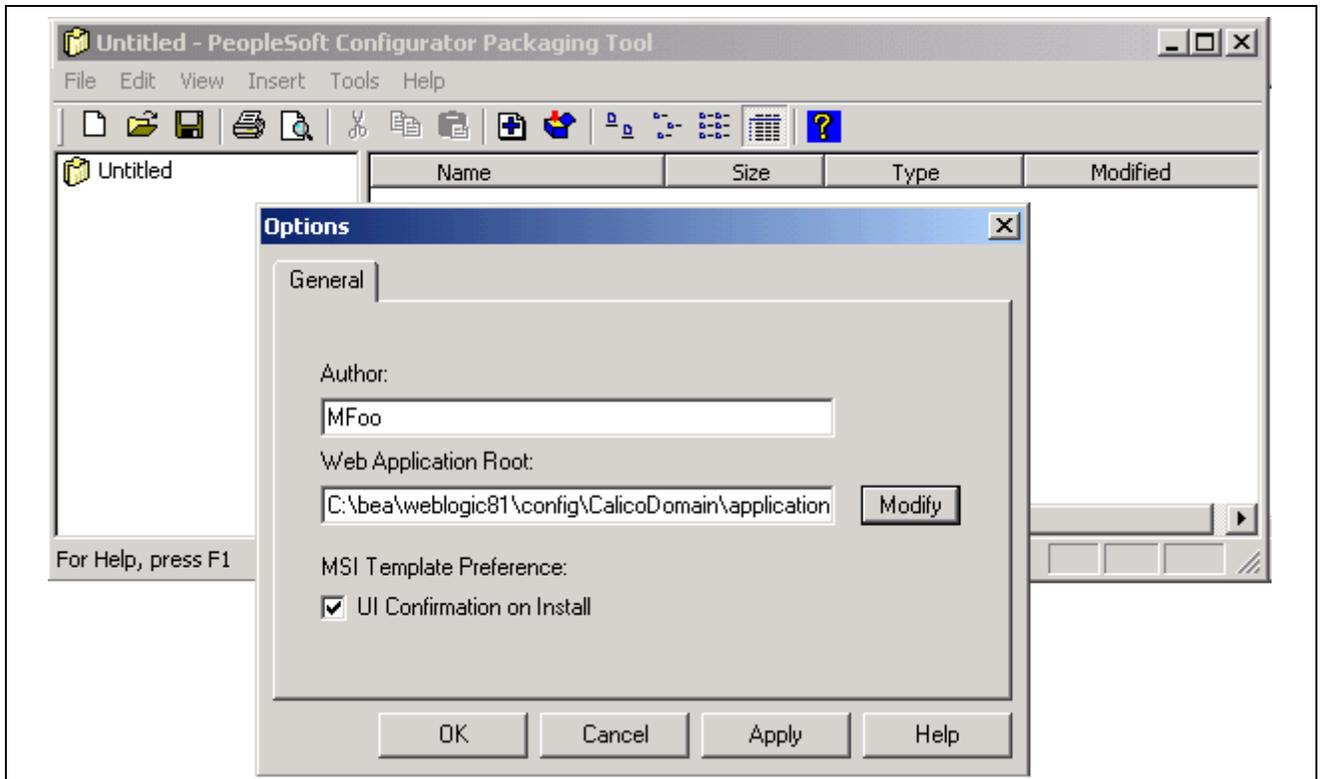
Packaging Tool main window

Setting the Application Root

The Packaging Tool must be pointed to the application root directory of the Configurator Server in order to create or modify a package. This information tells the tool where to find the available files for display in the Packaging Tool workspace.

If the Solution employs a compound model, rather than a component model, then the subdirectory containing the model's XML file is the root directory for the entire Solution, even if other parts of the Solution, including its component models, are located elsewhere. This separation of model location is possible because the compound model XML file contains the locations of its component models.

See [Chapter 30, "Understanding Mobile Product Configuration," Types of Models, page 390.](#)



Options dialog for specifying the location of the solution files

To set the application root:

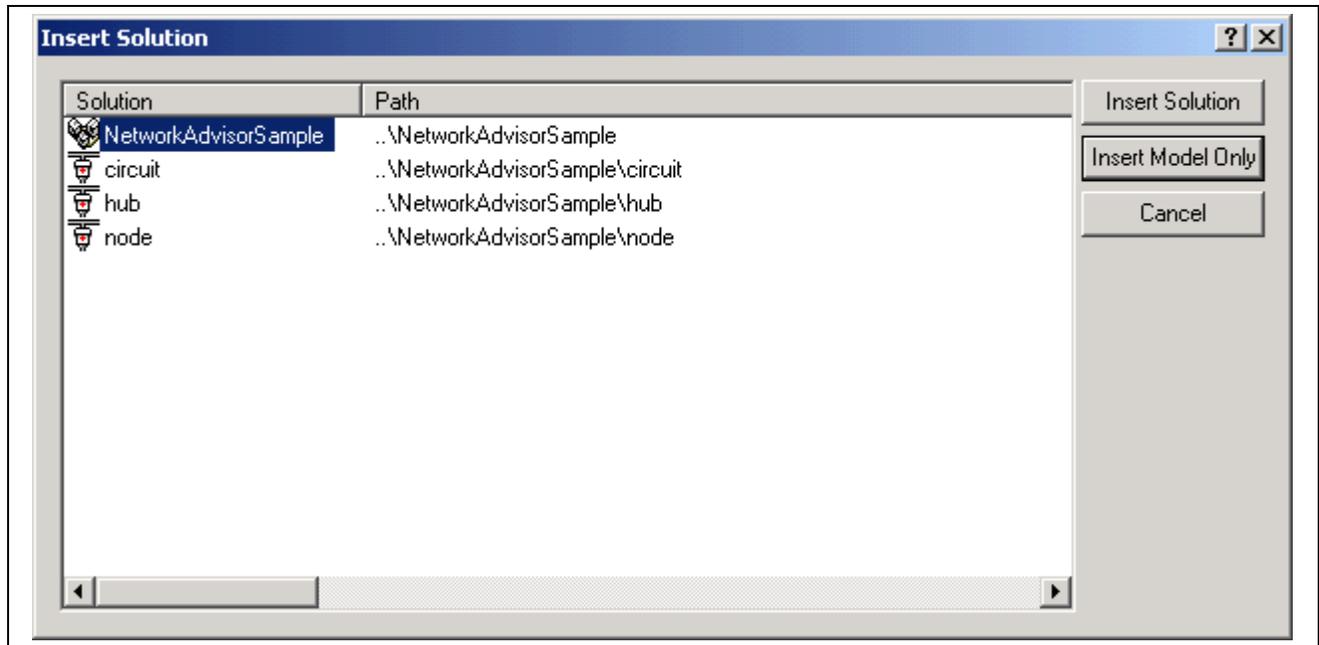
1. Select Tools, Options to display the Options dialog box:
2. (Optional) Designate an author (optional) and the location of the Solution files. This location should point to the folder containing the Configurator Solutions. This is typically the default web application root directory:
`C:\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\`
3. Click OK. The Packaging Tool is now ready to create new Packages.

Creating a New Solution Package

A Solution Package is the set of filenames for the files that you want included in the Solution Installer, which will be distributed to the mobile user. The Package has the extension.cci and contains filenames and references for those files.

There are three ways to insert application files into the Tool workspace for the creation of a Package:

- Insert the files of a Solution into a new Package.
- Open an existing Package and move, add, and remove files.
- Open an existing Package and insert another Package into it.



Selecting the solution to be packaged. Shown is a compound model Solution (selected) with three component models, Circuit, Hub, and Node

To create a Package from Solution files:

1. Set the application root.

See [Chapter 32, “Creating and Maintaining Mobile Solutions,” Setting the Application Root, page 404.](#)

2. Select File, New Package. The left pane displays the Package name at the top level of a hierarchy tree. The Package name, default “untitled”, is not editable in the pane; you can assign a name in the Properties dialog (right-click menu) or when you save it.
3. Select Insert, Solution. A dialog box displaying the Solutions in the specified web application root appears.

If the dialog lists no files, check that the root directory specified in the Options dialog points to the directory containing the application. The Packaging Tool looks for CalicoUI.properties and CalicoNA.properties in the specified root directory to locate Solutions to insert. If it does not find either of those files, the Insert Solution dialog will be blank or it will be missing part of a Solution.

4. Select the desired Solution and click Insert Solution.

Files from the application root appear in the tool workspace and are modifiable.

See [Chapter 32, “Creating and Maintaining Mobile Solutions,” Setting End-User Installation Options, page 407.](#)

5. Move, add, or remove the displayed files as needed and save the Package. You can now create its Solution Installer.

See [Chapter 32, “Creating and Maintaining Mobile Solutions,” Setting End-User Installation Options, page 407.](#)

To create a new Package from an existing Package:

Warning! It is possible to open a Package and insert another into it. However, this practice implies that you have already distributed an installer made from the inserted package already. When you distribute the installer for this merged package, there likely will be shared files for two applications on the mobile machine. Un-installing one or the other removes the shared file, thereby “breaking” the other application.

1. Select File, New Package. The left pane shows the top level, which is the Package name, initially “untitled.” The name is not editable in the pane. You can assign a name to the Package when it is saved, or you can right-click on “untitled,” select Properties, and edit the name.
2. Select Insert, Package to display the “Select Package to insert” dialog box.
3. Select the desired Package and click Open.

The Package panes of the Main Application window are then populated with the Package’s files.

4. Modify as needed.

See [Chapter 32, “Creating and Maintaining Mobile Solutions,” Setting End-User Installation Options, page 407.](#)

5. Save the Package. You can now create a Solution Installer.

See [Chapter 32, “Creating and Maintaining Mobile Solutions,” Setting End-User Installation Options, page 407.](#)

Adding a Model to the Solution Package

With the desired Package open, select Insert, Solution to display the Insert Solution dialog box, select the desired Solution, then click Insert Model Only.

A dialog box asks whether you want to delete the compiled version. Clicking Yes causes the compiled version to be overwritten.

The tool workspace will be populated with the model files. Unlike the operation of inserting a Solution, inserting a model restricts the inserted files to PeopleSoft Configurator model files only, omitting JSP pages, images, and other site-supporting files.

Setting End-User Installation Options

You can specify whether you want the end-user to have an “opt-out” message for package installation and to be able to uninstall the package.

To set end-user installation options”

1. Select Tools, Options to display the Options dialog box, then select the MSI Package Preferences tab.
2. If you want the mobile user to receive a dialog verifying their intention to install the Package—giving the mobile user a chance to opt out of the task—select the UI Confirmation on Install check box.
3. If you want to give the mobile user the ability to uninstall the package from their system’s Add/Remove Programs utility, select the Add Entry to Add/ Remove Programs check box.

4. Click Apply or OK.

Modifying a Solution Package

You can augment the application root files with non-Solution files, such as release documentation, for the mobile user.

Move, add, copy, and delete Package components in the main window. Moving, adding, copying, and deleting files conforms to Windows standards, with one exception: there is no drag-and-drop capability.

To add files to the Package, select File, New Folder for add directories, and Insert, Individual File(s) to add files.

To add new folders to the hierarchy, select File, New Folder.

Move, copy, and delete operations are windows-standard.

Adding Non-Solution Files to the Solution Package

The set of deploy files may not completely meet the needs of a mobile user environment. For instance, since you are ultimately preparing an installable package of files, a “release,” you may want to include documentation that addresses this release, with information specifically for the mobile user.

To add non-Solution files to the Package, select File, New Folder to add new directories, and Insert, Individual File(s) to insert files.

Accessing Package Properties

You can view the package’s properties in the Properties dialog. You can also edit the package name, title, and version. To access the Properties dialog, right-click on the Package name and choose Properties from the menu.

Adding Annotation to the Package

You can enter comments in the Properties dialog for a Package. To access the Properties box, right-click on the Package name and choose Properties from the menu, then click the Comments tab.

Generating the Solution Package Installer

The Solution Package installer, a .msi file, is the file that you will distribute to end users. The Package includes a Windows installer, to which you add to the solution package files.

The Solution Package Installer contains:

- Solution Package (.cci—a list of filenames and references).
- The model files, JSP pages, images, and other files required to run the Solution standalone on the mobile machine.
- Installation script for the above files.

To generate a Solution Package Installer, do *one* of the following:

- Select Tools, Create Solution Installer for a new package or for an upgrade that extends beyond a change in the model. You can also click its toolbar icon.

See [Chapter 32, “Creating and Maintaining Mobile Solutions,” Updating a Solution Package Installer, page 409.](#)

- Select Tools, Create Model Update Installer if you want to create a simple model upgrade containing the latest model version.

See [Chapter 32, “Creating and Maintaining Mobile Solutions,” Updating a Solution Package Installer, page 409.](#)

Updating a Solution Package Installer

Changes to the contents of a Solution Installer are made to its Solution Package and a new installer created. (File information in the installer file [.msi] is not easily editable.) Consequently, if you want to update the collection of files that you send to mobile users, you must re-open the associated package, make the desired modifications to file content, and generate a new Solution Installer from the updated Package.

Solutions contain model files, JSP pages, images, and any necessary support files. You can replace files with newer versions, add new files to the Package, and move and remove them. You can do the same with directory structures.

See Also

[Chapter 32, “Creating and Maintaining Mobile Solutions,” Adding a Model to the Solution Package, page 407](#)

[Chapter 32, “Creating and Maintaining Mobile Solutions,” Adding Non-Solution Files to the Solution Package, page 408](#)

Updating Only the Model(s) of a Package

Updates to a Solution often involve no more than rolling in the latest version of a model after a routine update to the product offerings, options, and attendant pricing and sales information. For this reason, the Packaging Tool offers a shortcut Package generation with a special menu command.

Note. A model-only update does not include updates to non-model files such as JSPs and images. In creating a Model Update, the Packaging Tool only changes what is in the installer.

See [Chapter 32, “Creating and Maintaining Mobile Solutions,” Setting End-User Installation Options, page 407.](#)

To generate a model-only update:

1. Open the Solution Package associated with the desired Solution release.
2. Select Insert, Solution. The Solution's newest model(s) are automatically retrieved and displayed in the Insert Solution dialog.
3. Select the desired model and click Insert Model Only.
4. Select Tools, Create Model Update Installer or click its icon on the tabular.

The Packaging Tool automatically locates the most recent version of the model and saves it.

A confirmation message appears. The new Model Update Installer does not overwrite the previous Solution Installer. It is saved automatically as <package name> Model Update.msi.

The new Solution Installer is designated <package name> Model Update.msi. It is saved in the same directory as the .cci files.

Printing Solution Package Content

To print a list of the files contained in a Solution Package, select File, Print.

PART 8

Advanced Configurator System Administration

Chapter 33
Understanding Advanced Configurator Administration

Chapter 34
Using Administration Tools

Chapter 35
Maintaining the Advanced Configurator System

CHAPTER 33

Understanding Advanced Configurator Administration

Advanced Configurator provides two tools to help with the development, testing, and deployment of a model:

- Administration Console
- Solution Tester

CHAPTER 34

Using Administration Tools

Designing and developing a model begins with the PeopleSoft Visual Modeler. The model defines selection points, domain members, and constraints. Domain members and constraints can be stored externally.

After you create the mode, you build the web site application using extensions for Dreamweaver, or more directly, using the PeopleSoft JSP templates for HTML form controls. Your web site application connects standard HTML form controls to your model.

At runtime, the web site sends user picks to the PeopleSoft Configurator engine through the Web Client Processor and the Client Operations Processor (COP). The engine processes the picks and returns the state of each item in the form controls to the web site. The engine also returns explanations for constraints that have been violated.

If the model has external domain members or constraints, the engine gets them from the Configuration Data Manager. If the model has pricing, the Client Operations Processor calculates the pricing information.

The diagram on the next page shows PeopleSoft Configurator's design, compile, and runtime environments.

Administration Console

PeopleSoft Configurator Administration console helps you develop, test, and deploy both component models and compound models.

The Administration console can internalize model data. It can import externally referenced model data and translate it into internally defined data. The model then becomes self-contained and can be deployed in a mobile configuration environment needing no access to the product database. The console can internalize data that is stored in a database as well as attribute list files.

On Windows NT, start the Administration tool by clicking Start and selecting Programs, PeopleSoft Applications, Configurator 9.0, Administration.

Alternatively, access the console directly from a browser with the uniform resource locator (URL) `http://<host>:<port>/ConfigServerAdmin`.

PeopleSoft.

Configuration Server Name:
localhost:7777
Configuration Server Version:
8.9.0.20040414

[Compound Structure List](#)

Compile Model XML File Browse...

Generate self-contained model. Additional Attribute File Browse...

Models	Available Operations
ATMFrameCircuit	Deploy Remove
8-9-0	Deploy Remove Update Data
20040420-140203-294	Deploy Disable Remove Run View(GCL RTP SGCL TMAP XML)
20040414-145843-587	Deploy Disable Remove Run View(GCL RTP SGCL TMAP XML)
20040329-104541-110	Deploy Disable Remove Run View(GCL RTP SGCL TMAP XML)
ATMFrameSite	Deploy Remove
8-9-0	Deploy Remove Update Data

Configurator Administration console

The Administration console enables you to do the following actions with an Advanced Configurator component model:

- **Compile**
You can compile a model that is anywhere on the network, using any server on the network. To compile, select the model's XML file, the server, and the port.
- **Run**
You can run the compile version of any model on the network. This launches the PeopleSoft Configurator Model Tester, which the PeopleSoft Visual Modeler also uses to test models. With the test client, you can submit the model with various options—such as pricing, auto-submission, sorting, and formatting—and reset the model.
- **Deploy**
You can deploy a model from anywhere on the network to any host server and port on the network. Deploying a model copies it to another machine for use there. You can deploy a compile version of a model, a major version of a model, or all versions of a model.
- **Disable**
You can disable a compile version of a model that is anywhere on the network. Disabling a model restricts its use to the Administration console, which can still run the model. If a model is disabled, the test client or a midtier application can't run it. Disabling a model does not delete it.
- **Enable**

If a model is disabled, you can enable it.

- Remove

You can remove—that is, delete—a model (including a compound model) from the network. You can remove a compile version of a model, a major version of a model, or all versions of a model.

Warning! During a model compile, the model version number is incremented based on the latest version of the model that is present on the server. Therefore, do not remove the latest version of a model before you recompile, because the new version may be given the version number of the deleted latest version.

Note. Removing the model from the server doesn't cause the model to be removed from server memory if it has been loaded into memory. The server must be restarted to force the model to be removed from memory. The model load settings for the server may also need to be changed.

- Update data

You can update any version of a model with model data (domain members and constraints) that is stored in an external database.

- Generate a self-contained model

If a model's data is stored externally, you can read that data into the model for internal storage and access. That compilation of the model then becomes self-contained and can be deployed in a configuration environment needing no access to the product database. You can also include additional attributes in the internalization process.

Note. The Visual Modeler can also internalize data; however, it can internalize only data that is used within the model. Externally referenced data that is not used in the model, such as descriptions, can't be internalized by the Visual Modeler.

Furthermore, the internalize processes in the Visual Modeler are a one-time effort. After a model is internalized in the Visual Modeler, the source is internal and all subsequent compilations produce versions that are internal as well.

- View

You can view the following types of files that are created in the compile version directory when a model is compiled: GCL, RTP, SGCL, TMAP, and XML. This lets you describe the state of these files if you need help.

In addition, the Administration console enables you to manage compound models:

- Upload

You can upload a compound model (in XML format) to the server.

- Cross-check

You can verify a compound model against its component models to verify that selection points and domain members that are required by the compound model exist.

- View

You can view a compound model in XML format.

Note. The Administration console does not provide security, but is based on industry standards for security. The information technologist and system administrator must provide and limit access to PeopleSoft Configurator servers on the network.

Testing Solutions

You can use the Solution Tester to develop and test Configurator solutions outside of the complete CRM environment.

The Solution Tester simulates the integrated CRM/Configurator environment, enabling you to test a solution without setting up configurable products or creating quotes or orders.

The Solution Tester enables you to test:

- Model operation.
- Page display.
- User interaction.
- Configuration pricing.
- Configuration details request and display.

It also provides access to parameters and details about the operation of the solution that aren't available in normal operation within the CRM environment. The Solution Tester runs completely within the Configurator Server.

The Solution Tester is organized into three sections. The top section contains a number of buttons that perform the operations that are provided by the Solution Tester. The middle section displays the results of the operations, including the user interface for a solution and the display of the configuration details for a particular solution configuration. The bottom section allows values to be specified that are passed into the Configurator solution when it is launched. These sections are described in detail in the following section.

Page Used to Test Solutions

The Solution Tester top panel controls the tester operations and displays the status that is returned from certain operations.

Access the Solution Tester page by selecting Start, Programs, PeopleSoft Applications, Configurator 9.0, Server Index Page. Then select Solution Tester.

Solution Tester page

New

Start a new configuration.

Clicking the New button starts a new configuration using the solution that was selected in the solution drop-down list. The solution's user interface is displayed in the center section of the Solution Tester. Interaction with the interface allows a new configuration to be created (see Configuring).

The New operation requires that a valid solution be selected in the solution drop-down list. The ConfigID and RetCode fields are cleared when you start a new configuration.

Restore

Restore an existing configuration.

Clicking the Restore button restores an existing configuration from the database. The solution's user interface appears in the center section of the Solution Tester with the current configuration values. Interaction with the interface modifies the current configuration (see Configuring).

The Restore operation requires that you select a valid solution in the Solution drop-down list and specify the Configuration ID for an existing (saved) configuration of the appropriate type in the ConfigID field. When you restore a configuration, the RetCode field is cleared.

Configuring

Configuring (from New or Restore).

Solutions that are designed to run within the integrated CRM/Advanced Configurator environment usually provide buttons that enable the user to exit the configuration without saving the current changes, or to save the current configuration and return (the labels on the buttons are defined in the user interface for the solution and cannot be Cancel and Return). Clicking these buttons on a solution's user interface within the Solution Tester initiates the same operations. Clicking Cancel returns the center section to the No Solutions Loaded state without saving the configuration. Clicking Return saves the current state of the configuration, updates the ConfigID and RetCode fields, and initiates the operation of retrieving and displaying the configuration details.

The status codes that are returned are:

- 0: Success.
- 101: Solution not found.
- 102: Invalid Config ID or Error restoring the configuration.
- 103: Error saving the configuration.
- 104: Error creating the configuration.

Copy

Copy an existing configuration.

Clicking the Copy button makes a copy of an existing configuration. A new configuration ID is returned and the ConfigID field is updated.

The Copy operation requires that you select a valid solution in the Solution drop-down list and specify the Configuration ID for an existing (saved) configuration of the appropriate solution type in the ConfigID field.

The status codes that are returned are:

- 0: Success.
- 101: Solution not found.
- 102: Invalid Config ID or Error restoring the configuration.
- 103: Error saving the configuration.
- 104: Error creating the configuration.

List Configurations

List the saved configurations. Clicking this button displays information about the configurations that is currently saved in the database. If you select a solution in the Solution drop-down list, only configurations of that type are displayed. If you don't select a solution, all of the saved configurations are displayed. Selecting a configuration from the list updates the Solution and ConfigID fields for use with the other operations.

View Details

View configuration details.

Clicking the View Details button requests, retrieves, and displays the returned configuration details for a configuration. The details are formatted using the style sheet that is defined for the solution.

For easy modification and testing, you can select the request (XML) and style sheet (XSLT) to use in the following way:

1. The Solution Tester checks the solution test directory (by default, C:\bea_cfg\weblogic81\config\CalicoDomain\applications\CalicoApp\calico\solutiontest) for a request XML file with the name <solutionname>.xml and for a style sheet XSLT file with the name <solutionname>.xslt (where solutionname is the name of the solution used for the configuration). If these are found, they are used for the request XML and the result formatting style sheet XSLT respectively.
2. If appropriate files are not found, the Solution Tester queries the database for the associated solution schema and attempts to retrieve the request XML and formatting style sheet XSLT that are defined by the schema.
3. If request XML and formatting XSLT cannot be found in the solution schema, or a solution schema cannot be found, the Solution Tester uses the default details request XML and formatting style sheet XSLT files in the solutiontest directory (ConfigDetailsRequest.xml and default.xslt).

The XML and XSLT data are read every time the details are requested. This sequence allows custom request XML and formatting XSLT to be tested, modified, and retested easily.

The View Details operation is automatically invoked when you return from a New or Restore operation using the return operation on the solution's user interface.

The View Details operation requires that you select a valid solution in the Solution drop-down list and specify the configuration ID for an existing (saved) configuration of the appropriate type in the ConfigID field.

View Details - XML

View request, response, and style sheet XML.

Clicking this button performs the same sequence of steps as the View Details button, except that the raw XML for the details request, details response, and formatting XSLT appears. Also provided is information indicating the source location and the size of the details request, and formatting XSLT. The three pieces of data are formatted into an XML document with the following structure:

```
<_details>
```

```

    <_request>
      [details request XML]
    </_request>
    <_response>
      [details response XML]
    </_response>
    <_stylesheet>
      [XSLT]
    </_stylesheet>
  </_details>

```

The View Details- XML operation requires that you select a valid solution in the Solution drop-down list and specify the configuration ID for an existing (saved) configuration of the appropriate type in the ConfigID field.

Save order

Change the order status of the configuration to *saved*. The Save Order operation requires that you select a valid solution in the Solution drop-down list and specify the configuration ID for an existing (saved) configuration of the appropriate type in the ConfigID field. When you save an order, the *RetCode* field is cleared.

Submit order

Change the order status of the configuration to *submitted*.

The Submit order operation requires that you select a valid solution in the Solution drop-down list and specify the configuration ID for an existing (saved) configuration of the appropriate type in the ConfigID field. When you submit an order, the RetCode field is cleared.

Cancel order

Roll back the current configuration to the state it was in when the order was last saved.

The Cancel order operation requires that you select a valid solution in the Solution drop-down list and specify the configuration ID for an existing (saved) configuration of the appropriate type in the ConfigID field. When you cancel an order, the RetCode field is cleared.

Delete order

Roll back the current configuration to the state it was in when the order was last submitted

The Delete order operation requires that you select a valid solution in the Solution drop-down list and specify the configuration ID for an existing (saved) configuration of the appropriate type in the ConfigID field. When you delete an order, the RetCode field is cleared.

Accessing the Solution Tester

To access the Solution Tester pages, select Start, Programs, PeopleSoft Applications, Configurator 9.0, Server Index Page.

From the Server Index Page, click Solution Tester.

Note. If “500 - Internal Server Error” appears rather than the tester page, verify that your database server is running and that the database entry for Advanced Configurator is valid. This information is entered during Configurator installation and stored in `\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\WEB-INF\config\JNDIDBName.properties`.

Solution Tester parameters

iFrame Height This parameter controls the height in pixels of the center (output) section of the Solution Tester.

Debug This sets a flag within the Solution Tester and passes a debug parameter to the Configurator solution (`debug=false` or `debug=true`). You can use the value of this parameter within a solution user interface to control debugging messages and other debugging or testing operations. When Debug is selected, the Solution Tester sends messages to the console window of the Configurator Server containing information about the operations that are being performed.

Understanding the Output and Solution User Interface

The large main panel displays the results of the operations, including the user interface for a solution when you are performing the New or Restore operations.

Because the main panel is an inline frame (iframe), it may contain an independent scroll bar, depending on the height of the content compared to the height of the iframe. When initiating an operation, you can set the height of the iframe using the iFrame Height field in the bottom section (the height cannot be changed during an operation).

The full URL that the Solution Tester uses to initially navigate the frame appears. This allows inspection of the values that are initially passed to the Configurator solution for New and Restore operations.

Every effort has been made to make the Solution Tester environment (the environment that is provided to the solution running within the tester) match that of the runtime environment of the CRM applications with which the Configurator integrates. However, some technical limitations prevent the two environments from being identical. Therefore, some differences might exist in operation between the Solution Tester and the actual CRM runtime environment.

In many cases, identifying and correcting these differences can help provide a better, more robust solution by identifying environment-specific functionality. You should carefully review differences to determine whether the solution is relying on functionality that may not be the same on every platform or environment.

In all cases, you should thoroughly test within the actual CRM runtime environment before deploying a new or updated solution.

Setting Configuration Solution Parameters

Scroll to the bottom panel of the Solution Tester page:

Property File - default -			
BusinessUnit	<input type="text" value="US001"/>	CaptureID	<input type="text" value="11111"/>
CurrencyCDBase	<input type="text" value="USD"/>	CustomerID	<input type="text" value="112233445566"/>
LineNbr	<input type="text" value="1"/>	Qty	<input type="text" value="1"/>
Date (YYYYMMDD)	<input type="text"/>	ProductID (blank = Solution ID)	<input type="text"/>
Channel	<input type="text" value="A"/>	UOM	<input type="text" value="EA"/>
CurrencyCD	<input type="text" value="USD"/>	KeepNew	<input type="text" value="TRUE"/>
DBName	<input type="text"/>	LanguageCD	<input type="text" value="ENG"/>
UserID	<input type="text" value="USER1"/>		
Additional URL Parameters <input type="text" value="moreParms=abc123"/>			
IFrame Height <input type="text" value="600"/>		IFrame Width <input type="text" value="800"/>	
<input type="checkbox"/> Debug			

Example Solution Tester page (2 of 2)

The bottom panel contains a number of fields that enable you to specify parameter values that are passed to the Configurator solution. The parameters that are passed are the same as those that are passed from the CRM applications when you are interfacing with the Configurator. Additional, user-defined parameters can also be passed.

You can modify the values for these parameters before performing operations. When an operation is initiated, changes to the parameter values do not take effect until you initiate a new operation. For instance, after you have started a new configuration by clicking the New button, changing the value of BusinessUnit has no effect until you start another operation, such as starting a new configuration or restoring an existing one.

The labels on the parameter value fields match the names of the parameters that are passed to the Configurator solution. The initial values for the fields are taken from a combination of values that are read from property files (text files in a specific format) and default values that are contained in the Solution Tester. The parameter fields are updated with values when you first open the Solution Tester or when you select a new property file in the Property File list.

The Solution Tester determines the values to use for each parameter in the following sequence:

1. It looks in the property file that is identified by the Property File list. If it finds a parameter=value pair for this parameter, it uses the value that is specified.
2. If it doesn't find an appropriate parameter entry, it looks in the default property file (SolutionTest.properties, in the solutiontest directory).
3. If it doesn't find a parameter entry in the default property file, it uses a default value that is defined within the Solution Tester.
4. If it doesn't find a default value, it leaves the field blank.

The Solution Tester populates the Property File list with the names of the property files that it finds in the solutiontest directory. The example property file ExternalChannel.properties is included with the installation. This file contains the following code:

```
# This file contains specific properties that should be used in place of
# those in the 'default' properties file (SolutionTest.properties)
#
# Channel can be 'A' (internal) or 'W' (external/web)
Channel=W
```

Notice that the only entry is for the parameter to override. All other values are read from the default property file or the Solution Tester defaults, or they are omitted. If you create a new property file while you are using the Solution Tester, you need to open a new instance of the Solution Tester for it to be added to the Property File list. The Additional URL Parameters field allows any number of additional parameters to be passed to the Configurator solution.

Enter additional parameters in the following way:

- To pass a single additional parameter, enter: `parameter=value`.
- To pass more than one parameter, enter: `parameter1=value1¶meter2=value2¶meter3=value3`.

Model Tester

Configurator also includes a web-based interface that you launch from the Visual Modeler with the Run command.

The Model Tester provides a standard, preformatted, front-end UI for testing a model's relationships, error message display, expression output, and control specifications. Using it, the modeler can perform the actions of the end user, making selections and entering data so as to observe actual model behavior at runtime.

For you to use the Model Tester, the Configurator Server must be installed and running at the time of its use.

CHAPTER 35

Maintaining the Advanced Configurator System

This chapter discusses how to:

- Manage model versions.
- Load models.
- Manage the memory usage of the Configurator server.
- Compress configuration data.
- Use the Explanations.properties file.
- Compile models from the command line.
- Access and use COPXML servlet statistics.

Managing Model Versioning

Different versions of the same model and multiple instances of the same version can coexist. Models have a three-part version number plus a compile version number.

The compile version number contains the date and time. For example:

```
20040320-184840-135
```

This example version number indicates that the model was compiled at 6:48 p.m. on March 20, 2004.

The version number is separated into parts by hyphens. The parts are the major, minor, and subminor versions of the model. The modeler assigns the major and minor version numbers in the PeopleSoft Visual Modeler. The compiler assigns the subminor version number. It performs a cyclical redundancy check and advances the subminor version number if the model changes. An example is 0-1-3.

This example indicates that the modeler assigned a version number of 0-1, and that the compiler has detected three changes in the model since it was first compiled as version 0-1-0.

Note. The following description assumes that you accepted default settings when you installed the PeopleSoft Configurator server.

All models are compiled in subdirectories of the PeopleSoft Configurator models directory. Subdirectories exist for the model, each part of the version number, and the compile version number. For example:

```
<weblogic home>\weblogic81\config\CalicoDomain\applications\CalicoApp\WEB-INF
\models\myBestModel
  0
    1
      0
```

20000306-063520-356

The compile version directory contains all the files that are created during linking and compiling. The parent directories have no files, although you can move the Explanations.properties file to them. Each time the model is compiled, the compiler creates a new compile directory and all the files in it. It creates a subminor directory only if the model changes.

The compiler creates the following files in the compile version directory:

- Explanations.properties
- Gcl
- Lep
- Map
- Rtp
- Sgcl
- Tmap
- Xml

Loading Models

You can specify which models the PeopleSoft Configurator server loads at startup by setting one or two parameters for the Configurator's startup servlet in the WebLogic Administrative console.

To specify the Configurator server settings:

1. Open the WebLogic Administrative console in a browser using its domain address `http://<server>:<port>/console`.
2. Navigate to CalicoDomain, Deployments, Web Application, CalicoApp.
3. Click Edit Web Application Descriptor.

A new browser window appears where the parameters can be set.

4. In the new window, navigate to Web Descriptor, Web App Descriptor, Servlets, StartupServlet, Parameters.

The default is a single parameter with name=load and value=default. The load parameter can have one of these values:

- *All*, which loads every model in the models\ subdirectory.
 - *Default*, which loads the latest compile version of the latest version of each model in the models\ subdirectory.
 - *Specific*, which requires the second parameter, models.
5. Add the models parameter:
 - a. Click Configure a new Parameter
 - b. Set the name to *models*.
 - c. Specify a value using one of the following approaches:

Specify at least one model name and its version and compile version. If you specify only the model name, the application server loads the latest compile version of the latest version of the model that you specify.

or

Specify the model name and one of the following versions or combinations:

major version

major and minor version

major, minor, and sub-minor version

major, minor, sub-minor, and compile version (which fully specifies the model)

Example:

Version specification	Action taken
load=all	Loads all compile versions of all models in the models directory.
load=default	Loads the latest compile version of the latest version of all models in the models directory.
load=specific,models= myModelOne:0-1-0:20000306- 185244-355;myModelTwo; myModelThree:0-2	Loads the specified compile version of myModelOne, the latest compile version of the latest version of myModelTwo, and the latest compile version of the latest subminor version of minor version 2 of major version 0 of myModelThree.

6. Navigate to the top level of the tree in the left-hand pane (called Web Descriptor), and click on the Persist button.
7. Restart the server.

Managing the Memory Usage of the Configurator Server

When a model is compiled on the Configurator server, it is translated from its XML representation into a set of object instances in the Java Virtual Machine (JVM). This set of object instances is then serialized to an RTP (runtime problem) disk file. When a client later requests the model, this file is used to reconstruct the needed object instances (hereafter referred to as a ModelSpec instance). Because reconstructing a ModelSpec from the RTP file significantly affects performance, ModelSpec instances are cached in memory.

To prevent the JVM from exhausting its available memory as new versions of models are added to the server, the administration servlet removes older ModelSpec instances from the cache. Administrators can control the cache size with the `model.cache.size` property in the `Advisor.properties` file:

```
model.cache.size=20
```

If not specified, the property's value is *10* versions by default.

To minimize the performance impact of a cache size setting that is too small, the cache comprises a primary and secondary cache. The cache size setting refers to the size of the primary cache alone. The secondary cache acts as a holding bin for the oldest ModelSpec instances, and they are subject to the regular deletion cycles of the JVM.

The primary cache contains the most recently used ModelSpec instances. As ModelSpec instances are requested from the cache, they are added to the primary cache if they are not already there. If the primary cache has reached its maximum size, then the least recently used ModelSpec instance in the primary cache is pushed to the secondary cache.

Compressing Configuration Data

PeopleSoft Configurator enables you to compress configuration xml data. Any compressed data is decompressed during a restore operation when the configuration is requested.

Two modes of data compression are available:

- Compression at runtime with the user's save request.
- Compression for maintenance of existing configuration data stores.

Use the maintenance compression mode to compress configuration data that has not been compressed during runtime saves.

To specify data compression at runtime:

1. Locate the `Advisor.properties` file in

```
\\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\WEB-INF\config
```

2. Set the property `calico.na.db.compression` equal to `true` if you want to enable data compression.

The `calico.na.db.compression` property enables and disables compression for the saved configurations of all solutions that are deployed on the web application server. Its default setting is `true`.

To compress configuration data on existing databases, run the following code from a command line. A utility compresses the saved configurations on the specified database.

```
java -classpath <CONFIGURATOR_JAR_PATH>;<CONFIGURATOR_CONFIG_PATH>;
<DATABASE_DRIVER_PATHS> calico.cms.persistence.DBUtil -compress
```

where

<CONFIGURATOR_JAR_PATH> is the path to the Configurator jar file (`advisor.jar`).

<CONFIGURATOR_CONFIG_PATH> is the path to the Configurator properties files.

<DATABASE_DRIVER_PATHS> is the path to both of the database drivers.

Example:

```
java -classpath r:\MKTG_DEN_CRM\cfg\advisor.jar;r:\CRM_VOB\pscfg\3rdparty\jdbc
=>
\classes12.zip;r:\MKTG_DEN_CRM\cfg\jdbc\mssqldrivers.zip;E:\bea\weblogic81
\config\CalicoDomain\applications\CalicoApp\WEB-INF
\config=>
calico.cms.persistence.DBUtil -compress
```

Using the Explanations.properties File

The Explanations.properties file contains messages about constraint violations. The compiler creates an Explanations.properties file in each compile version directory. This file contains key-value pairs that describe constraint violations, which are displayed in the test client and in the midtier application. Each explanation is keyed by its constraint name. For example:

```
LeatherSeatsSedanCompatibility=Leather seats are not available with this vehicle.
WhiteMeatMustardCatsupCompatibility=White meat is not compatible with
mustard or⇒
    ketchup. The condiment you selected is not available with
the meat you selected.⇒
    ModemToMotherBoardDynDef=An external drive
(floppy or CD-ROM) is recommended.⇒
    FramesToTintDynDef=Constraint
4.B.2 RadioMultipleCDCDChangerRequired=The multiple⇒
    CD radio
requires a CD changer.
```

Copying the Explanations.properties File

You can move or copy the Explanations.properties file to the following directories:

- <ModelName>

Placing the Explanations.properties in the <ModelName> directory applies it to all versions and compile versions of that model, unless an Explanations.properties file is also in a child directory, that is, in a <MajorVersion>, <MinorVersion>, or <CompileVersion> directory for the model. The file in a child directory overrides the file in the parent (<ModelName>) directory.

- <MajorVersion>

Placing the Explanations.properties in the <MajorVersion> directory applies it to all minor versions and compile versions of that major version of the model, unless an Explanations.properties file is also in a child directory, that is, in a <MinorVersion> or <CompileVersion> directory for that major version of the model. The file in any child directory overrides the file in any parent (<MajorVersion> or <ModelName>) directory.

- <MinorVersion>

Placing the Explanations.properties in the <MinorVersion> directory applies it to all compile versions of that minor version of the model, unless an Explanations.properties file is also in a <CompileVersion> directory for that minor version of the model. The file in a <CompileVersion> directory overrides the file in any parent (<MinorVersion>, <MajorVersion>, or <ModelName>) directory.

Also, you can place an Explanations.properties file in the models directory. It applies to all compile versions of all models in the models directory. For example:

```
COMPLETENESS_CONSTRAINT=You must provide a selection for this choice.
COMPATIBILITY_CONSTRAINT=The selections that you made are not compatible.
```

Searching for the Explanations.properties File

The Configurator test client that is used in the PeopleSoft Visual Modeler searches through directories for the Explanations.properties file in the following order:

1. <MinorVersion>
2. <MajorVersion>
3. \models\ (the root directory for all <ModelName>s)
4. <ModelName>
5. <CompileVersion>

You can design your JavaServer Pages (JSP) midtier application to search in another order.

Compiling Models from the Command Line

If Configurator implementation requires regular or frequent model updates, you can save time and effort by using the command-line compile utility. This compile command calls the executable for the Visual Modeler, which opens briefly to compile the .csw files that you supply in the command. It also generates a log file.

The command is the Visual Modeler executable, `-compile`, and the full or relative path of the model that you want to compile:

```
ClicViM -compile mymodel.csw
```

To compile multiple models, call each separately.

The models can be local or remotely located; however, the filepath must be to a machine on which the Visual Modeler is installed. If you want, you can store the compiled files to an internet protocol (IP) address.

Accessing and Using COPXML Servlet Statistics

The COPXML servlet tracks various processing times for interactive xml requests:

- Hits: The total number of requests.
- Overall time(ms): The time to process a request.
- Start time(ms): The time that is spent parsing the request.
- Init time(ms): The time that is spent initializing the COP with the correct model and version.
- Execute time(ms): the time that is spent processing the request (creating choices and processing them through the engine).
- Extract tTime(ms).
- Generate time(ms): The time that is spent generating the response XML.
- Max simultaneous: The maximum number of simultaneous requests.

This is an example:

```
?@COPXMLServlet Statistics
```

```
@Hits 10 Total number of requests
```

```
@Overall Time(ms) 631 63 avg Total time to process request
```

@Start Time(ms) 380 38 avg Time spent parsing the request

@Init Time(ms) 90 9 avg Time spent initializing the cop with the correct model/version

@Execute Time(ms) 61 6 avg Time spent processing the request. (Creating choices and processing them through the engine)

@Extract Time(ms) 0 0 avg Not used

@Generate Time(ms) 100 10 avg Time spent generating the response XML

@Max Simultaneous 1 Maximum number of simultaneous requests

Note. All times are for processing interactive requests only. Times processing ConfigDetails, ConfigCopy, or Mobile(Compound) requests are not included in the statistics with the exception of Start Time, which records the parsing of all requests. The first number in each row is the accumulated time since the server was started. The second number is the average per request.

To view the COPXML statistics, navigate to the copxml servlet in a browser window, using, for instance, <http://localhost:7777/copxml>

To reset the values without restarting the server, use the following post URL:

<http://localhost:7777/copxml?reset=true>

APPENDIX A

Visual Modeler Expression Editor Functions

The following table describes the functions and operators that are provided for creating expressions.

Tables are:

- Numeric operators and functions
- Boolean functions
- Date functions
- String functions

Numeric Operators and Functions

The Visual Modeler supports these operators and functions:

Function	Description	Example
-	Subtract.	$x - y$
%	Return the remainder of x divided by y.	$x \% y$
*	Multiply.	$x * y$
**	Return x raised to the power y.	$x ** y$
/	Divide.	x / y
+	Add.	$x + y$
<	Less Than.	$x < y$
<=	Less Than or Equal To.	$x <= y$
<>	Not Equal.	$x <> y$
=	Equal.	$x = y$
>	Greater Than.	$x > y$
>=	Greater Than or Equal To.	$x >= y$
abs()	Return the absolute value of x.	abs(x)

Function	Description	Example
acos()	Return the arccosine of the radian value <i>x</i> .	acos(<i>x</i>)
addDays()	Add the integer value (1-31) of the day of a given date <i>x</i> .	addDays(<i>x</i>)
addMonths()	Add the integer value (1-12) of the month of a given date <i>x</i> .	addMonths(<i>x</i>)
addYears()	Add the integer value of the year of a given date <i>x</i> .	addYears(<i>x</i>)
asin()	Return the arcsine of the radian value <i>x</i> .	asin(<i>x</i>)
avgWithQty()	Total the value of all picks divided by the number of discrete items picked.	avgWithQty(<i>x</i> , <i>y</i>)
bnd()	Return the first argument that has a bound value. Unlike other functions, it will only propagate its value if one of the arguments is bound; thus it can be used to prevent automatic propagation of selection point attribute references. Note that if the last value is a constant, then that will be returned as the default value. So <code>bnd(sp1, sp2, 4)</code> returns 4 if neither <i>sp1</i> nor <i>sp2</i> are bound.	bnd(<i>x</i> 1,..., <i>x</i> n)
compareTo()	Compare value <i>x</i> to value <i>y</i> and return an integer value: < 0 if value <i>x</i> is less than value <i>y</i> , 0 if value <i>x</i> equals value <i>y</i> , and > 0 if value <i>x</i> is greater than value <i>y</i> .	compareTo(<i>x</i> , <i>y</i>)
concatenate()	Concatenate object values <i>x</i> 1 through <i>x</i> n into a string.	concatenate(<i>x</i> 1,..., <i>x</i> n)
cos()	Return the cosine of the radian value <i>x</i> .	cos(<i>x</i>)
cot()	Return the cotangent of the radian value <i>x</i> .	cot(<i>x</i>)
countWithQty()	Return the number of discrete items. For example, if the picks are drive (quantity of 3), cpu (1), and monitor (1), the count with quantity is 5.	countWithQty(<i>x</i> , <i>y</i> , <i>z</i>)
date()	Return a date constructed from integer year <i>y</i> , month <i>m</i> , and day <i>d</i> .	date(<i>y</i> , <i>m</i> , <i>d</i>)
dateToInt()	Convert the date <i>x</i> to an integer in the form YYYYMMDD.	dateToInt(<i>x</i>)

Function	Description	Example
daysBetween()	Return the number of days (always positive) between dates x and y. If x and y are the same days, zero is returned.	daysBetween(x,y)
doesNotEqual()	Return a boolean value indicating whether value x has the same type and value as value y.	doesNotEqual(x,y)
equals()	Return a boolean value indicating whether value x has the same type and value as value y.	equals(x,y)
getDay()	Return the integer value (1-31) of the day of a given date x.	getDay(x)
getMonth()	Return the integer value (1-12) of the month of a given date x.	getMonth(x)
getYear()	Return the integer value of the year of a given date x.	getYear(x)
if()	Compare values and return the first output if the comparison is True, and the second output if False.	if(x>y,Out1,Out2)
indexOf()	Return an integer indicating the position of integer x within number y optionally beginning at integer position z. -1 is returned if integer x is not found within number y.	indexOf(x,y,z)
intToDate()	Convert the integer x to a date in the form YYYYMMDD.	intToDate(x)
length()	Return the length in characters of number value x.	length(x)
max()	Return the value of the largest argument	max(x,y,z)
maxWithQty()	Multiply the value for each attribute by quantity, if any, then return the largest value.	maxWithQty(x,y,z)
min()	Return the smallest argument.	min(x,y,z)
minWithQty()	Multiply each argument by quantity, if any, then return the smallest value.	minWithQty(x,y,z)
pi()	Return the value of pi.	pi()
product()	Multiply arguments.	product(x,y,z)

Function	Description	Example
quotient()	Return the integer result of x / y .	quotient(x,y)
round()	Return the value x rounded to integer precision y . If y is greater than 0, whole number rounding is performed, otherwise decimal rounding is performed. For example, round(126.456,1) is 126 and round(126.456,-1) is 126.5.	round(x,y)
sin()	Return the sine of the radian value x .	sin(x)
sqrt()	Return the square root of x .	sqrt(x)
substring()	Return the substring value of x starting at position y optionally ending at position z if specified.	substring(x,y,z)
sum()	Add arguments.	sum(x,y,z)
sumWithQty()	For each argument, multiply quantity (if used) times the attribute value and add the results.	sumWithQty(x,y,z)
tan()	Return the tangent of the radian value x .	tan(x)
toDegrees()	Return the radian value x converted to degrees.	toDegrees(x)
toFloat()	Convert a float, integer, string, or boolean value to a float.	toFloat(x)
toInteger()	Convert a float, integer, string, or boolean value to an integer.	toInteger(x)
toRadians()	Return the degree value x converted to radians.	toRadians(x)

Boolean Functions

The following table list the boolean functions provided for building expressions.

Function	Description	Sample
!	not()	NOT
&	and()	Logical AND
^	xor()	Exclusive OR

Function	Description	Sample
	or()	Logical OR
<	Less Than	$x < y$
<=	Less Than or Equal To	$x \leq y$
<>	Not Equal	$x \neq y$
=	Equal	$x = y$
>	Greater Than	$x > y$
>=	Greater Than or Equal To	$x \geq y$
and()	Returns true if all the inputs are true.	and(x,y,z)
bnd()	Returns the first argument that has a bound value. Unlike other functions, it will only propagate its value if one of the arguments is bound; thus it can be used to prevent automatic propagation of selection point attribute references. Note that if the last value is a constant, then that will be returned as the default value. So <code>bnd(sp1, sp2, 4)</code> returns 4 if neither <code>sp1</code> nor <code>sp2</code> are bound.	bnd(x1,...,xn)
compareTo()	Compares value x to value y and returns an integer value: < 0 if value x is less than value y, 0 if value x equals value y, and > 0 if value x is greater than value y.	compareTo(x,y)
concatenate()	Concatenates object values x1 through xn into a string.	concatenate(x1,...,xn)
contains()	Returns a boolean value indicating whether string x contains string y.	contains(x,y)
countWithQty()	Returns the number of discrete items. For example, if the picks are drive (quantity of 3), cpu (1), and monitor (1), the count with quantity is 5.	countWithQty(x,y,z)
doesNotContain()	Returns a boolean value indicating whether string x does not contain string y.	doesNotContain(x,y)
doesNotEqual()	Returns a boolean value indicating whether value x has the same type and value as value y.	doesNotEqual(x,y)
endsWith()	Returns a boolean value indicating whether string x ends with string y.	endsWith(x,y)

Function	Description	Sample
equals()	Returns a boolean value indicating whether value x has the same type and value as value y.	equals(x,y)
if()	Compare values and return the first output if the comparison is True, and the second output if False.	if(x>y,Out1,Out2)
not()	Returns negated input value.	not(x)
occursAfter()	Returns the boolean value true if date x occurs after date y, otherwise the value false is returned.	occursAfter(x,y)
occursOnOrAfter()	Returns the boolean value true if date x occurs on or after date y, otherwise the value false is returned.	occursOnOrAfter(x,y)
occursOnOrBefore()	Returns the boolean value true if date x occurs on or before date y, otherwise the value false is returned.	occursOnOrBefore(x,y)
or()	Returns true if any input is true.	or(x,y,z)
sortsAfter()	Returns a boolean value indicating whether string x sorts after string y using a Collator object based on the current locale.	sortsAfter(x,y)
sortsBefore()	Returns a boolean value indicating whether string x sorts before string y using a Collator object based on the current locale.	sortsBefore(x,y)
startsWith()	Returns a boolean value indicating whether string x starts with string y.	startsWith(x,y)
toFloat()	Converts a float, integer, string, or boolean value to a float.	toFloat(x)
toInteger()	Converts a float, integer, string, or boolean value to an integer.	toInteger(x)
xor()	Returns true if only one input value is true.	xor(x,y,z)

Date Functions

The Visual Modeler supports these date functions:

Function	Description	Sample
addDays()	Adds y days to date x returning the new date.	addDays(x,y)
addMonths()	Adds y months to date x returning the new date.	addMonths(x,y)
addYears()	Adds y years to date x returning the new date.	addYears(x,y)
bnd()	Returns the first argument that has a bound value. Unlike other functions, it will only propagate its value if one of the arguments is bound; thus it can be used to prevent automatic propagation of selection point attribute references.	bnd(x1,...,xn)
compareTo()	Compares value x to value y and returns an integer value: < 0 if value x is less than value y, 0 if value x equals value y, and > 0 if value x is greater than value y.	compareTo(x,y)
concatenate()	Concatenates object values x1 through xn into a string.	concatenate(x1,...,xn)
countWithQty()	Returns the number of discrete items. For example, if the picks are drive (quantity of 3), cpu (1), and monitor (1), the count with quantity is 5.	countWithQty(x,y,z)
date()	Returns a date constructed from integer year y, month m, and day d.	date(y,m,d)
dateToInt()	Converts the date x to an integer in the form YYYYMMDD.	dateToInt(x)
daysBetween()	Returns the number of days (always positive) between dates x and y. If x and y are the same days, zero is returned.	daysBetween(x,y)
doesNotEqual()	Returns a boolean value indicating whether value x has the same type and value as value y.	doesNotEqual(x,y)
equals()	Returns a boolean value indicating whether value x has the same type and value as value y.	equals(x,y)
getBeginningOfMonth()	Returns the date of the beginning of the month (the 1st) for a given date x.	getBeginningOfMonth(x)
getBeginningofWeek()	Returns the date of the closest Monday on or before a given date x.	getBeginningofWeek(x)

Function	Description	Sample
getBeginningofYear()	Returns the date of the beginning of the year for a given date x. The first day of the year is considered to be January 1st.	getBeginningofYear(x)
getDay()	Returns the integer value (1-31) of the day of a given date x.	getDay(x)
getMonth()	Returns the integer value (1-12) of the month of a given date x.	getMonth(x)
getSolveDate()	Returns the date passed to the PSProblemState solve method (which is also the date used to calculate effectivity).	getSolveDate(x)
getToday()	Returns the current date	getToday(x)
getYear()	Returns the integer value of the year of a given date x.	getYear(x)
if()	Compare values and return the first output if the comparison is True, and the second output if False.	if(x>y,Out1,Out2)
intToDate()	Converts the integer x to a date with the YYYYMMDD format.	intToDate(x)
occursAfter()	Returns the boolean value true if date x occurs after date y, otherwise the value false is returned.	occursAfter(x,y)
occursBefore()	Returns the boolean value true if date x occurs before date y, otherwise false is returned.	occursBefore(x,y)
occursOnOrAfter()	Returns the boolean value true if date x occurs on or after date y, otherwise the value false is returned.	occursOnOrAfter(x,y)
occursOnOrBefore()	Returns the boolean value true if date x occurs on or before date y, otherwise the value false is returned.	occursOnOrBefore(x,y)
toDate()	Converts the string value x to a date. Format is YYYY-MM-DD.	toDate(x)
toFloat()	Converts a float, integer, String, or boolean value to a float.	toFloat(x)
toInteger()	Converts a float, integer, String, or boolean value to an integer.	toInteger(x)

String Functions

The following table describes the string functions available for creating expressions.

Function	Description	Sample
bnd()	Returns the first argument that has a bound value. Unlike other functions, it will only propagate its value if one of the arguments is bound; thus it can be used to prevent automatic propagation of selection point attribute references.	bnd(x1,...,xn)
compareTo()	Compares value x to value y and returns an integer value: < 0 if value x is less than value y, 0 if value x equals value y, and > 0 if value x is greater than value y.	compareTo(x,y)
concatenate()	Concatenates object values x1 through xn into a string.	concatenate(x1,...,xn)
contains()	Returns a boolean value indicating whether string x contains string y.	contains(x,y)
countWithQty()	Returns the number of discrete items. For example, if the picks are drive (quantity of 3), cpu (1), and monitor (1), the count with quantity is 5.	countWithQty(x,y,z)
doesNotContain()	Returns a boolean value indicating whether string x does not contain string y.	doesNotContain(x,y)
doesNotEqual()	Returns a boolean value indicating whether value x has the same type and value as value y.	doesNotEqual(x,y)
endsWith()	Returns a boolean value indicating whether string x ends with string y.	endsWith(x,y)
equals()	Returns a boolean value indicating whether value x has the same type and value as value y.	equals(x,y)
if()	Compare values and return the first output if the comparison is True, and the second output if False.	if(x>y,Out1,Out2)
indexOf()	Returns an integer indicating the position of string x within string y optionally beginning at integer position z. -1 is returned if string x is not found within string y.	indexOf(x,y,z)

Function	Description	Sample
length()	Returns the length in characters of string value x.	length(x)
sortsAfter()	Returns a boolean value indicating whether string x sorts after string y using a Collator object based on the current locale.	sortsAfter(x,y)
sortsBefore()	Returns a boolean value indicating whether string x sorts before string y using a Collator object based on the current locale.	sortsBefore(x,y)
startsWith()	Returns a boolean value indicating whether string x starts with string y.	startsWith(x,y)
substring()	Returns the substring value of x starting at position y optionally ending at position z if specified.	substring(x,y,z)
toDate()	Converts the string value x to a date. Format is YYYY-MM-DD.	toDate(x)
toFloat()	Converts a float, integer, String, or boolean value to a float.	toFloat(x)
toInteger()	Converts a float, integer, String, or boolean value to an integer.	toInteger(x)
toLowerCase()	Returns a string in which all upper case letters in string x have been converted to lower-case letters.	toLowerCase(x)
toUpperCase()	Returns a string in which all lower case letters in string x have been converted to upper-case letters.	toUpperCase(x)
trim()	Returns a string in which all white space from both ends of string x has been removed.	trim(x)

APPENDIX B

Creating and Adding User-Defined Functions

This appendix discusses the use of user-defined functions.

User-defined functions allow model developers to extend the expression capabilities of PeopleSoft Advanced Configurator to meet specific needs not addressed by its pre-defined functions. This appendix provides instructions and information to help you add your own function.

In addition, a sample user-defined function called `getQuantity` is provided with the Configurator install. It consists of the compiled class file and Java source file, which are located in: `\samples\Configurator\SampleSolutions\GetQuantity_UDF\classes`. Also included is a model and a readme file describing the sample function and how to add it to your system for demonstration. You can also modify the sample source file to create your own function.

Adding a User-Defined Function

To add a user-defined function:

1. Stop the Configurator Server if it is running.
2. Create a Java source file that implements the `UserFunction` and `java.io.Serializable` interfaces. To implement the `UserFunction` interface, add a method to the class that calculates the function's return value. A number of functions are provided to access the arguments passed to the `calculate` method.

See [Appendix B, "Creating and Adding User-Defined Functions," Implementing the UserFunction Interface, page 444.](#)

3. Compile this source file.
4. Copy the newly compiled class file into the `\classes` directory of `\\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\Web-inf`. You may need to create the `\classes` directory.
5. Locate `UserFunctions.xml` in `\\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\Web-inf\config`.
6. Add the following lines between the `<FUNCTION_LIST>` tags:

```
<FUNCTION NAME="getQuantity" CATEGORY="user" RETURN_TYPE="float"  
CLASS=>  
"Get=>QuantityFunction"  
1#####DEFAULT_ARGUMENT_TYPE="integer" DEFAULT_ARGUMENT_NAME="getQuantity-var"  
  
1#####MIN_ARGUMENTS="2" MAX_ARGUMENTS="2"/>
```

See [Appendix B, “Creating and Adding User-Defined Functions,” Editing UserFunctions.xml, page 446.](#)

The Visual Modeler queries the server or other supported source for the contents of the UserFunctions.xml file to determine the names of the functions to be displayed in the expression editor.

At compile time, the compiler employs the UserFunctions.xml file to detect the user-defined functions.

7. Save the file and restart the Configuration Server.
8. In the expression editor of the Visual Modeler, click the “Refresh Functions from Server” button. Visual Modeler queries the Configurator server for the contents of the UserFunctions.xml.
9. Compile the model to complete the process. The Visual Modeler connects to the Configurator Server, reads the UserFunctions.xml file, and updates the editor’s list of user-defined functions available for modeling.

Implementing the UserFunction Interface

The primary method in the UserFunction interface is **calculate()** and must be defined by any class implementing the UserFunction interface.

The **calculate()** method is used to compute the result of the user-defined function, which is returned as an Object instance. Within the **calculate()** method, the function arguments can be retrieved from the *UserFunctionsArgument* instance using the methods provided by its class.

Methods

The UserFunction interface provides these methods:

numberOfArguments()	Returns the number of arguments that were passed into the user function. Each individual argument passed into the user function is actually a list of values. In the case of constants, return values from other functions, and single-select selection point attributes, the list will contain only one value. In the case of multiselect selection point attributes, the list can contain multiple values.
argumentContainsMultipleValues(int argumentPosition)	Returns true if an argument comprises multiple values, otherwise false is returned. As for Java arrays, references to the position of an argument begin at 0.
numberOfArgumentValues(int argumentPosition)	Returns the number of values associated with the specified argument position. If called for an argument for which argumentContainsMultipleValues() would return false, this method returns 1.

<p>argumentIsInteger(int argumentPosition) argumentIsInteger(int argumentPosition, int valuePosition) argumentIsDouble(int argumentPosition) argumentIsDouble(int argumentPosition, int valuePosition) argumentIsNumber(int argumentPosition) argumentIsNumber(int argumentPosition, int valuePosition) argumentIsBoolean(int argumentPosition) argumentIsBoolean(int argumentPosition, int valuePosition) argumentIsString(int argumentPosition) argumentIsString(int argumentPosition, int valuePosition)</p>	<p>Used to determine the type of an argument value. These methods take either one argument, the argument position, or two arguments, the argument position and value position in the argument list. If only the argument position is specified, the value position is assumed to be 0. The return value of these methods is either true, if the value is of the specified type, or false, if the value is not of the specified type.</p>
<p>argumentIntegerValue(int argumentPosition) argumentIntegerValue(int argumentPosition, int valuePosition) argumentDoubleValue(int argumentPosition) argumentDoubleValue(int argumentPosition, int valuePosition) argumentBooleanValue(int argumentPosition) argumentBooleanValue(int argumentPosition, int valuePosition) argumentStringValue(int argumentPosition) argumentStringValue(int argumentPosition, int valuePosition)</p>	<p>Used to retrieve an argument value. These methods take either one argument, the argument position, or two arguments, the argument position and value position in the argument list. If only the argument position is specified, the value position is assumed to be 0. The return value of these methods is the specified value from the argument list. The method <code>argumentIntegerValue()</code> will coerce a float value to an integer value and <code>argumentDoubleValue()</code> will coerce an integer value to a float value.</p>
<p>argumentQuantity(int argumentPosition); argumentQuantity(int argumentPosition, int valuePosition);</p>	<p>Used to retrieve the quantity associated with argument values. In the case of constants and return values from other functions, the quantity returned is always 1. In the case of selection point attributes, the quantity returned is the quantity associated with the selected domain member that is associated with the attribute value.</p>
<p>argumentLongValue(int argumentPosition, int valuePosition) argumentLongValue(int argumentPosition);</p>	<p>These methods allow a user-defined function to request arguments as long integer values rather than integer values. These methods behave the same as the existing <code>argumentIntegerValue</code> methods except that they return a Java long rather than a Java int. Call these methods if you expect to retrieve numbers larger than 2,096,000,000 (a Java-imposed limit to the int type).</p> <p>If you don't, and such a huge number gets entered into your UDF, it will get truncated (and might become negative or some other funny effect)</p>

Exceptions

If the argument or value positions are invalid or if an invalid type is requested, the **UserFunctionException** exception will be thrown by the argument access methods.

Editing UserFunctions.xml

Entries in UserFunctions.xml must contain either:

- The name of the Java class that implements the user-defined function (the NAME tag), or
- The name of the function as displayed by the Visual Modeler and referenced in expressions.

Or,

You can specify:

- The return type of the function (the RETURN_TYPE tag).
- The minimum and maximum number of arguments expected by the function (the MIN_ARGUMENTS and MAX_ARGUMENTS tags).
- The default argument type (the DEFAULT_ARGUMENT_TYPE tag).
- The default argument aggregation (the DEFAULT_ARGUMENT_AGGREGATION tag).
- The default argument name (the DEFAULT_ARGUMENT_NAME).

Tag definitions are:

RETURN_TYPE	Legal values are integer, float, double, number, string, date, boolean, long, and object. If this tag is unspecified, it is assumed to be object. Using number as the return type signifies that the return value will be either a double or integer. Using float as the return type is identical to using double as the return type. Using object as the return type signifies that the return value will be either integer, double, string, date, or boolean.
MIN_ARGUMENTS and MAX_ARGUMENTS	Should be integers or the symbol “variable”. If unspecified, these tags are assumed to be variable.
DEFAULT_ARGUMENT_TYPE	Has the same allowed types as the RETURN_TYPE tag.
DEFAULT_ARGUMENT_AGGREGATION	Should be a Boolean value. By default it is assumed to be false.
DEFAULT_ARGUMENT_NAMED	Is a description of the function’s arguments.

Specify information for each argument using the ARGUMENT tag.

For each argument, specify the NAME tag needs to be specified for each argument and is a description of the argument.

The type can be specified for each argument using the TYPE tag and aggregation can be specified using the AGGREGATOR tag. The TYPE tag has the same allowed types as the RETURN_TYPE tag. By default, the type is object.

The AGGREGATOR tag should be a boolean value and by default is false. If an argument is specified to be an aggregator, then the compiler enforces the restriction that only selection point/attribute references, function calls, and numeric variables can be specified for this argument in an expression.

Examples

The content portion of an example UserFunctions.xml file illustrates the use of the second method above:

```

public class TripleFunction
    implements UserFunction, java.io.Serializable
    {
        public Object calculate( UserFunctionArguments args)
            {
                try
                {
                    if (args.argumentIsInteger(0))
                        { return new Integer(3 * args.argumentIntegerValue(0)); }
                    else if (args.argumentIsDouble(0))
                        { return new Double(3.0 * args.argumentDoubleValue(0)); }

                    return(new Integer(0));
                }
                catch (UserFunctionException e)
                {
                    return(new Integer(0));
                }
            }
    }

```

The following example is of a user function called **tripleWithQty**, which takes 1 or more numeric arguments. Each argument value is multiplied by 3 and its associated quantity and the resulting values are summed and returned as a Double value.

```

import com.calico.engine.config.lightning.compiler.UserFunction;
import com.calico.engine.config.lightning.compiler.UserFunctionArguments;
import com.calico.engine.config.lightning.compiler.UserFunctionException;

public class TripleWithQtyFunction
    implements UserFunction, java.io.Serializable
    {
        public Object calculate( UserFunctionArguments args)
            {
                try
                {
                    int i, j;
                    double sum = 0.0;

                    for (i = 0; i < args.numberOfArguments(); i++)
                    {
                        for (j = 0; j < args.numberOfArgumentValues(i); j++)
                        {
                            if (args.argumentIsNumber(i,j))
                            {
                                sum += 3.0 * args.argumentDoubleValue(i,j) *
                                    args.argumentQuantity(i,j);
                            }
                        }
                    }
                }
            }
    }

```

```

        return(new Double (sum));
    }
    catch (UserFunctionException e)
    {
        return(new Double(0.0));
    }
}
}

```

Using the Sample User-Defined Function `getQuantity()`

This section describes the sample user-defined function called `getQuantity()` that is provided with the Visual Modeler.

- Setting up `getQuantity()`.
- Viewing `getQuantity()` behavior.

Understanding the `getQuantity()` Sample Function

`getQuantity()` is a sample user-defined function that returns the selected quantity of a specific domain member within a multi-select selection point. The associated files and a sample model implementing this function are included on the Configurator CD. You can use this sample in your own modeling or simply to understand how to define and implement your own expression functions.

Suppose you want to know the quantity of the selected domain member in a single-selection point. You could use the pre-defined function `countWithQty()` to retrieve that value. However, this function cannot recognize more than one selected domain member, so you must use the `getQuantity()` sample function for multi-select selection points.

The sample `getQuantity()` user-defined function requires two files and a sample model to demonstrate its use:

- `GetQuantityFunction.java` Java user-defined function (for source use).
- `GetQuantityFunction.class` compiled Java user-defined function.
- `UserFunctions.xml` user-defined XML for `getQuantity()`.

The compiled class file and Java source file are located on the PeopleSoft Advanced Configurator CD at:

```

\samples\Configurator\SampleSolutions\GetQuantity_UDF\classes

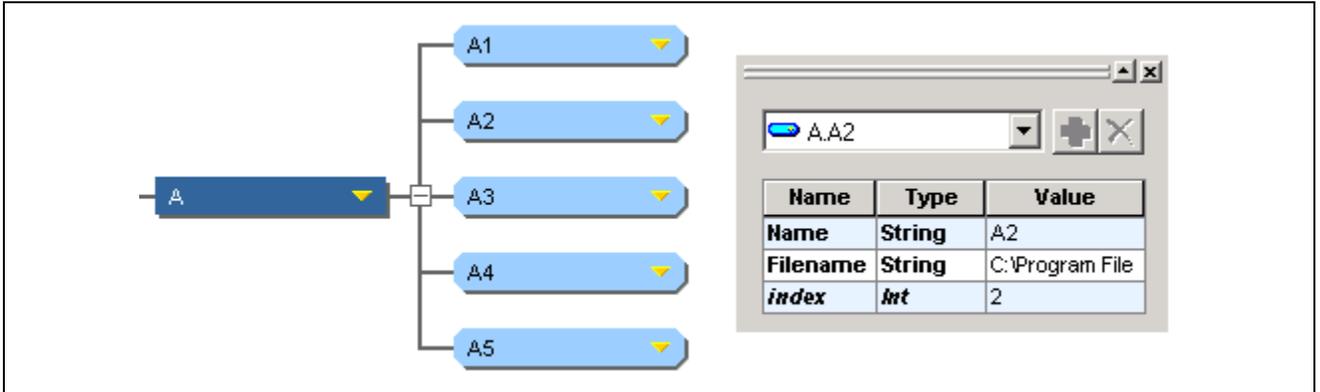
```

The `getQuantity()` function has two arguments:

- An integer representing an integer attribute value on the specific domain member holding the desired quantity.
- The Selection Point/attribute name combination for the Selection Point containing the domain member holding the desired quantity.

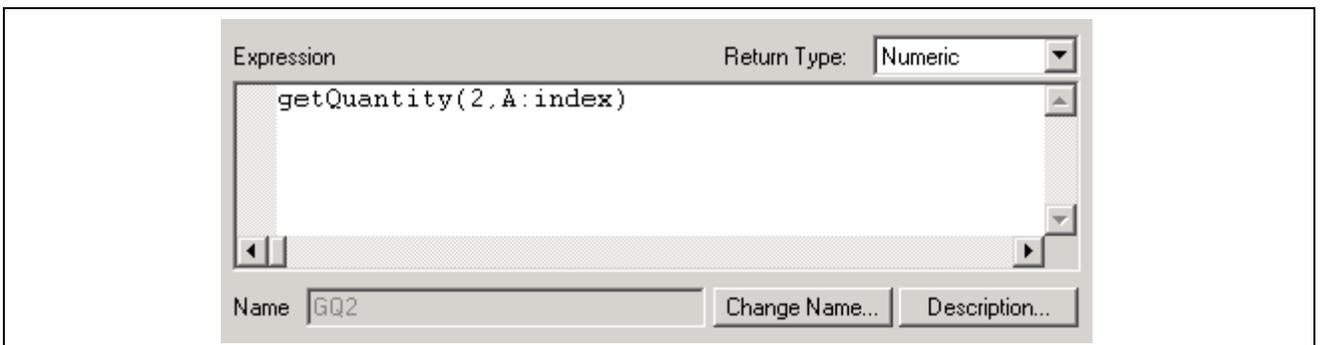
In order for the function to evaluate properly, the target domain member must have an attribute value equal to the same value of the first argument.

In the figure below, the domain member A2 has an attribute called *index* with an integer value of 2. If you wished to obtain the quantity of the A2 domain member, the first argument in the `getQuantity()` function would be 2.



Partial Structure View and expression Properties Editor for domain member A2

Note. Set the return type for `getQuantity()` to *Numeric*. The expression in the figure below, named GQ2 in the sample model, will return the quantity selected on the A2 domain member.



Expression editor for `getQuantity()`

Setting Up `getQuantity()`

To demonstrate `getQuantity()`, you must first place it and the model files in the appropriate directories.

To install the function:

1. Stop the PeopleSoft Advanced Configurator Server if it is running.
2. Create a `\classes` folder that resides within the `\Web-inf` folder of your PeopleSoft Advanced Configurator installation. For example:

```
C:\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\Web-Inf
```

3. Copy the `getQuantityFunction.class` file to the `\classes` folder just created.
4. Locate `UserFunctions.xml` on the Configurator CD:

```
\samples\Configurator\SampleSolutions\GetQuantity_UDF\WEB-INF\config
```

5. Do *one* of the following:

If no user-defined functions have been installed in your PeopleSoft Advanced Configurator environment, you can copy the entire `UserFunctions.xml` file into the `config` folder, located typically in:

```
C:\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\
Web-inf\config
```

If user-defined functions are already installed in your PeopleSoft Advanced Configurator environment, open the UserFunctions.xml file from the CD and copy and paste the following text to your existing UserFunctions.xml file between the <FUNCTION_LIST> tags:

```
<FUNCTION NAME="getQuantity" CATEGORY="user" RETURN_TYPE="float"
CLASS="GetQuantityFunction"
DEFAULT_ARGUMENT_TYPE="integer"
DEFAULT_ARGUMENT_NAME="getQuantity-var"
MIN_ARGUMENTS="2" MAX_ARGUMENTS="2"/>
```

6. Locate the getQuantity sample model on the Configurator CD in:

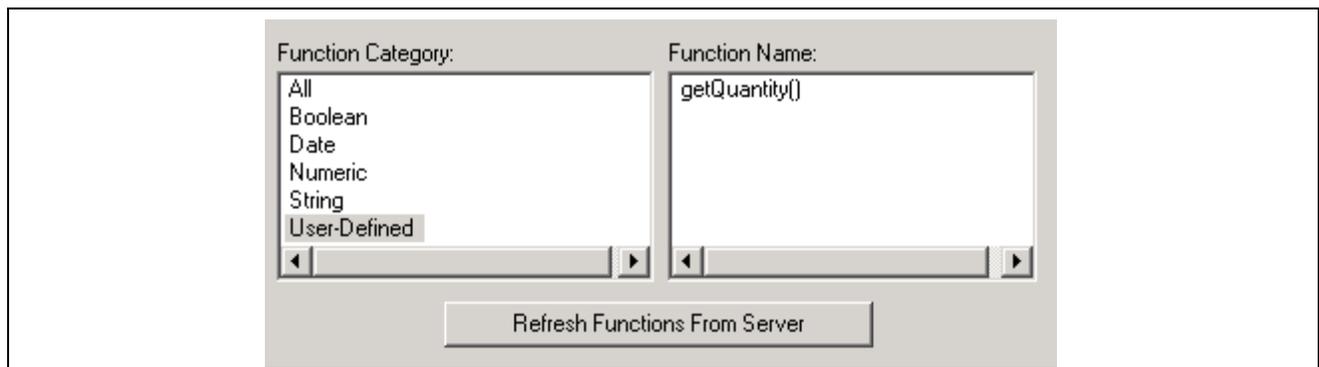
```
\samples\Configurator\SampleSolutions\GetQuantity_UDF\model
```

7. Copy the getQuantity folder and its contents to:

```
C:\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\
Web-inf\classes
```

Note. Do not place the getQuantity folder and its contents in a subfolder.

8. Start the PeopleSoft Advanced Configurator Server.
9. Start the Visual Modeler and open the getQuantity sample model you just copied.
10. Select Project, Settings to verify that the model is pointing at the correct hostname for your Configurator Server.
11. Open any expression (for example, GQ2) by double-clicking on one in the Components windows.
12. Click on the Refresh Functions From Server button. The Visual Modeler will connect to the Configurator Server, read the UserFunctions.xml file, and update the list of user-defined functions available for modeling.
13. Click on the compile icon in the toolbar of the Visual Modeler. Verify that the model was compiled successfully by viewing its name in the Visual Modeler message window, as shown:

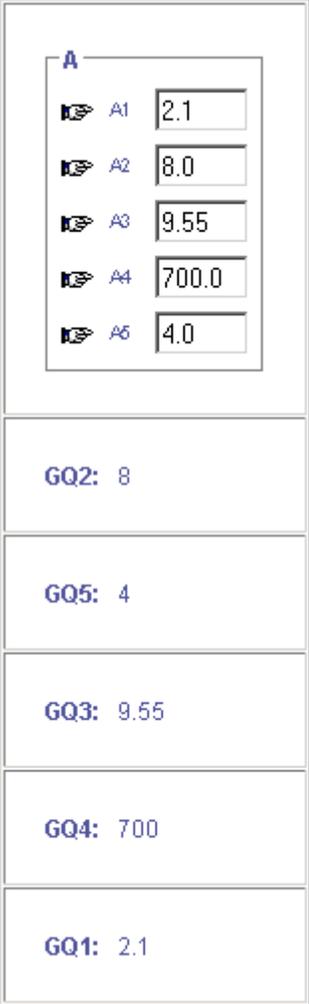


Updated function list

Viewing getQuantity() Behavior

This section discusses how to run the GetQuantity model so as to observe results from the getQuantity() function.

To see the runtime behavior of `getQuantity()`, compile and run the `getQuantity()` sample model by clicking the Compile and run the model icon in the toolbar of the Visual Modeler. Enter quantity values for each domain member on the selection point and click the Submit button. Note that the G1-G5 expressions return the quantity values entered. Also note that the GQ2 expression returns the quantity value of A2, as it was used in the examples above.



Domain Member	Quantity Value
A1	2.1
A2	8.0
A3	9.55
A4	700.0
A6	4.0

GQ2	8
GQ5	4
GQ3	9.55
GQ4	700
GQ1	2.1

Model Tester showing `getQuantity()` runtime behavior

APPENDIX C

Advanced Configurator Form Controls

This appendix provides the specific code for the form controls supported by Advanced Configurator for use in developing custom user interfaces:

- Single-Select Group Form Control
- Multi-Select Group Form Control
- Single-Select List Form Control (Drop-Down)
- Multi-Select List Form Control
- Single-Select Table Form Control
- Multi-Select Table Form Control
- Single-Select Image Table
- Application Why Help
- Form Control Why Help
- Text Input Form Control
- Numeric Data Form Control
- Extern Entry

Single-Select Group Form Control

The following JSP code generates a single-select group form control in HTML:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);
    params.put(PARAM_ATTRIBUTES, <Attributes>);
    params.put(PARAM_CONTROLID, <Control ID>);
    params.put(PARAM_CAPTION, <Caption>);
    params.put(PARAM_GENERATENONEMODE, <Generate None Mode>);
    params.put(PARAM_GENERATENONETEXT, <Generate None Text>);
    generateSingleSelectGroup(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

Note. The form control template filenames are in mixed case, i.e. SingleSelectGroup.jsp. For the deployments running on Solaris systems, make sure the characters of the filename match correctly (case-sensitive) with the template filenames.

The following is the HTML output of the previous JSP code:

```
<FONT COLOR="ctrlStateFontColor">Caption</FONT>
<IMG BORDER="0" SRC="ctrlStateImage"><BR>
<IMG ALT="ControlItemStateTag" BORDER="0" SRC="ControlItemStateImage">
<INPUT TYPE="radio" NAME="Control ID" VALUE="$NADA"><FONT COLOR="...">GenerateNone=>
Text&nbsp;</FONT><BR>
<IMG ALT="- " BORDER="0" SRC="itemStateImag">
<INPUT TYPE="radio" NAME="Control ID" VALUE="ControlItemName~State">><FONT COLOR=>
"...">
```

Multi-Select Group Form Control

The following JSP code creates a multiple-select group form control in HTML.

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);
    params.put(PARAM_ATTRIBUTES, <Attributes>);
    params.put(PARAM_CONTROLID, <Control ID>);
    params.put(PARAM_CAPTION, <Caption>);
    generateMultiSelectGroup(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

The following is the HTML output of the previous JSP code:

```
<FONT COLOR="ctrlStateFontColor">Caption</FONT>

<IMG BORDER="0" SRC="ctrlStateImage"><BR>

<IMG ALT="ControlItemStateTag" BORDER="0" SRC="ControlItemStateImag">

<INPUT TYPE="checkbox" NAME="Control ID" VALUE="ControlItemName~State">><FONT=>
COLOR="...">

ControlItemName DeltaPrice</FONT><BR>
```

Single-Select Table Form Control

The following JSP code creates a single-select table form control in HTML:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);
    params.put(PARAM_ATTRIBUTES, <Attributes>);
    params.put(PARAM_CONTROLID, <Control ID>);
    params.put(PARAM_CAPTIONIMAGE, <Caption Image>);
    params.put(PARAM_COLUMNHEADINGS, <Column Headings>);
    params.put(PARAM_GENERATENONEMODE, <Generate None Mode>);
    params.put(PARAM_GENERATENONETEXT, <Generate None Text>);
    generateSingleSelectTable(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endsparan -->
```

The following is the HTML output of the previous JSP code.

```
<FONT COLOR="ctrlStateFontColor"> Caption </FONT>
<IMG BORDER="0" SRC="ctrlStateImage"><BR>
<TABLE>
  <TR>
    <TH></TH>
    <TH>ColumnHeadingItem</TH>
    ...
  </TR>
  <TR>
    <TD><IMG ALT="itemStateTag" SRC="itemStateImage">
      <INPUT TYPE="radio" NAME="Control ID" VALUE="$NADA">
    </TD>
    <TD><FONT COLOR="ctrlItemStateFontColor">GenerateNoneText</FONT>
    </TD>
  </TR>
  <TR>
    <TD><IMG ALT="-" BORDER="0" SRC="itemStateImage">
      <INPUT TYPE="radio" NAME="Control ID"
        VALUE="ControlItemName~ItemState">
    </TD>
    <TD><FONT COLOR="ctrlItemStateFontColor">ControlItemAttribute
      </FONT>
    </TD>
    ...
  </TR>
  ...
</TABLE>
```

Multi-Select List Form Control

The following JSP code creates a multiple-select list form control in HTML.

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);
    params.put(PARAM_ATTRIBUTES, <Attributes>);
    params.put(PARAM_CONTROLID, <Control ID>);
    params.put(PARAM_CAPTION, <Caption>);
    generateMultiSelectList(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

The following is the HTML output of the previous JSP code.

```
<FONT COLOR="ctrlStateFontColor"> Caption </FONT>
<IMG BORDER="0" SRC="ctrlStateImage"><BR>
<SELECT NAME="Control ID " MULTIPLE >
<OPTION VALUE="$NADA">GenerateNoneText
<OPTION VALUE="ControlItemName~State" >StateTag ControlItemName DeltaPrice
```

Single-Select List Form Control

The following JSP code creates a single-select list form control in HTML:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);
    params.put(PARAM_ATTRIBUTES, <Attributes>);
    params.put(PARAM_CONTROLID, <Control ID>);
    params.put(PARAM_CONTROLSIZE, <Control Size>);
    params.put(PARAM_CAPTION, <Caption> );
    params.put(PARAM_GENERATENONEMODE, <Generate None Mode>);
    params.put(PARAM_GENERATENONETEXT, <Generate None Text>);
    generateSingleSelectList(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

The following is the HTML output of the previous JSP code.

```
<FONT COLOR="ctrlStateFontColor"> Caption </FONT>
<IMG BORDER="0" SRC="ctrlStateImage"><BR>
<SELECT NAME="Control ID ">
<OPTION VALUE="$NADA">GenerateNoneText
<OPTION VALUE="ControlItemName~State" >StateTag ControlItemName DeltaPrice
```

Multi-Select Table Form Control

The following JSP code creates a multiple-select table form control in HTML:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
if (params != null)
{
params.clear();
params.put(PARAM_OBJECTNAME, <Object Name>);
params.put(PARAM_ATTRIBUTES, <Attributes>);
params.put(PARAM_CONTROLID, <Control ID>);
params.put(PARAM_COLUMNHEADINGS, <Column Headings>);
params.put(PARAM_CAPTION, <Caption>);
generateMultiSelectTable(params, request, out);
}
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

The following is the HTML output of the previous JSP code:

```
<FONT COLOR="ctrlStateFontColor"> Caption </FONT>
<IMG BORDER="0" SRC="ctrlStateImage"><BR>
<TABLE>
<TR>
<TH></TH>
<TH>ColumnHeadingItem</TH>
...
</TR>
<TR>
<TD><IMG ALT="itemStateTag" BORDER="0" SRC="itemStateImage">
<INPUT TYPE="checkbox" NAME="Control ID"
VALUE="ControlItemName~ItemState " >
</TD>
<TD><FONT COLOR="ctrlItemStateFontColor">ControlItemAttribute
</FONT>
</TD>
...
</TR>
...
</TABLE>
```

Single-Select Image

Pass parameters to the template, and create a single-select image on the model's Web page as follows:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);
    params.put(PARAM_DOMAINMEMBERNAME, <Domain Member Name>);

    params.put(PARAM_IMAGEATTRIBUTE, <Image Attribute>);
    //or
    //params.put(PARAM_IMAGENAME, <Image Name>);

    //optional image parameters
    params.put(PARAM_IMAGEPATH, <Image Path>);
    params.put(PARAM_IMAGEWIDTH, <Image Width>);
    params.put(PARAM_IMAGEHEIGHT, <Image Height>);
    params.put(PARAM_IMAGESTATES, <Image States>);
    params.put(PARAM_MOUSEOVERIMAGES, <Mouseover Images>);
    params.put(PARAM_ALTTEXTATTRIBUTE, <Alt Text Attribute>);
    params.put(PARAM_ALTTEXT, <Alt Text>);
    params.put(PARAM_ADDITIONALATTRIBUTES, <Additional Attributes>);
    params.put(PARAM_AUTOSUBMIT, <Autosubmit>);

    //optional callback parameters
    params.put(PARAM_CREATECALLBACK, <Create Callback>);
    params.put(PARAM_SELECTCALLBACK, <Select Callback>);
    params.put(PARAM_UNSELECTCALLBACK, <Unselect Callback>);
    params.put(PARAM_MOUSEOVERCALLBACK, <Mouseover Callback>);
    params.put(PARAM_MOUSEOUTCALLBACK, <Mouseout Callback>);
    params.put(PARAM_AUTOSUBMITCALLBACK, <Autosubmit Callback>);

    generateSingleSelectImage(params, request, out);
  }
  %>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

Single-Select Image Table

Pass parameters to the template and create a single-select image table on the model's Web page as follows:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
```

```

    {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);

    params.put(PARAM_CAPTION, <Caption>);
    //or
    //params.put(PARAM_CAPTIONIMAGE, <Caption Image>);

    params.put(PARAM_IMAGEATTRIBUTE, <Image Attribute>);

    //optional standard Advisor parameters
    params.put(PARAM_SORT, <Sort>);
    params.put(PARAM_FILTERELIMINATEDITEMS, <Caption Image>);
    params.put(PARAM_FILTERELIMINATIONLEVEL_LOWER, <Lower E Level>);
    params.put(PARAM_FILTERELIMINATIONLEVEL_UPPER, <Upper E Level>);
    params.put(PARAM_COMPARATOR, <Comparator>);

    //optional image parameters
    params.put(PARAM_IMAGEPATH, <Image Path>);
    params.put(PARAM_IMAGEWIDTH, <Image Width>);
    params.put(PARAM_IMAGEHEIGHT, <Image Height>);
    params.put(PARAM_IMAGESTATES, <Image States>);
    params.put(PARAM_MOUSEOVERIMAGES, <Mouseover Images>);
    params.put(PARAM_ALTTEXTATTRIBUTE, <Alt Text Attribute>);
    params.put(PARAM_ADDITIONALATTRIBUTES, <Additional Attributes>);
    params.put(PARAM_AUTOSUBMIT, <Autosubmit>);

    //optional callback parameters
    params.put(PARAM_CREATECALLBACK, <Create Callback>);
    params.put(PARAM_SELECTCALLBACK, <Select Callback>);
    params.put(PARAM_UNSELECTCALLBACK, <Unselect Callback>);
    params.put(PARAM_MOUSEOVERCALLBACK, <Mouseover Callback>);
    params.put(PARAM_MOUSEOUTCALLBACK, <Mouseout Callback>);
    params.put(PARAM_AUTOSUBMITCALLBACK, <Autosubmit Callback>);

    //optional table parameters
    params.put(PARAM_COLUMNS, <Columns>);
    params.put(PARAM_ROWS, <Rows>);
    params.put(PARAM_BORDER, <Border>);
    params.put(PARAM_CELLSPACING, <Cell Spacing>);
    params.put(PARAM_CELLPADDING, <Cell Padding>);

    generateSingleSelectImageTable(params, request, out);
    }
    %>
<!-- METADATA TYPE="CalicoControl" ends span -->

```

Application Why Help

The following JSP code creates the application why help in HTML:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_CAPTION, <Caption>);
    generateApplicationWhyHelp(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

The Caption parameter value is the string to be displayed as the caption for the violations. The Caption Image is the image to be displayed as the caption for the violations. Specify either Caption or Caption Image but not both.

Note. There are no required parameters for the Application Why Help form control template.

The following is the HTML output of the previous JSP code:

```
Caption
<LI>ViolationItem 1
<LI>ViolationItem 2
```

Form Control Why Help

The following JSP code creates the Form Control Why Help in HTML:

```
<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);
    params.put(PARAM_CAPTION, <Caption>);
    generateControlWhyHelp(params, request, out);
  }
%>
<!-- METADATA TYPE="CalicoControl" endspan -->
```

Note. PARAM_OBJECTNAME is the only required parameter for the Control Why Help form control template.

The following is the HTML output of the previous JSP code:

```
Caption
```

```

<LI>ViolationItem 1
<LI>ViolationItem 2
...

```

Text Input Form Control

Pass parameters to the template, and create the text input text box on the model's Web page as follows:

```

<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);

    params.put(PARAM_CAPTION, <Caption>);
    //or
    //params.put(PARAM_CAPTIONIMAGE, <Caption Image>);

    // optional parameters
    params.put(PARAM_TEXTINPUTDEFAULT, <Default Entry>);
    params.put(PARAM_ENTRYSIZE, <Entry Size>);

    generateTextInput(params, request, out);
  }
  %>
<!-- METADATA TYPE="CalicoControl" endspan -->

```

Numeric Data Form Control

The following JSP code creates a numeric data control on the model's web page:

```

<!-- METADATA TYPE="CalicoControl" startspan-->
<%
  if (params != null)
  {
    params.clear();
    params.put(PARAM_OBJECTNAME, <Object Name>);

    params.put(PARAM_CAPTION, <Caption>);
    //or you could use
    //params.put(PARAM_CAPTIONIMAGE, <Caption Image>);
    // but not both caption parameters.

```


APPENDIX D

Compound Model Properties File

Before you can test or deploy a compound model, you must specify its operating properties in `CalicoNA.properties`. This file is located in `\\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\solutions\CompoundSample`.

This appendix provides an explanation of the properties and the text of the file.

Properties Description

This section explains each property and the values expected as they appear in the file. A complete, non-annotated version of the file follows.

```
# Auto Save On Session Timeout (true/false)
calico.session.autosave=false
```

If set to true, this property will save any configurations that are open when the session times out. It is recommended that you consider all the ramifications of this behavior before setting the flag to true.

```
# Solution info
calico.solution.name=TelcoDemo
calico.solution.version=1-2
calico.solution.description=Component Modeling Demo
calico.solution.allowNew=true
```

These properties are read and displayed by the solutions list servlet.

```
calico.page.restore=
```

See [Chapter 23, “Understanding the Runtime System,” Restore Policy, page 303](#).

This property is reserved for features to be added in a future release. Leave it at its default value.

```
# Compound Structure File
calico.compoundstructure.name=TelcoDemo
calico.compoundstructure.version=1-0
calico.compoundstructure.restore.policy=1
```

These properties define the Compound Structure Document used when using compound configurations. Default is the `TelcoDemo` (`\TelcoSample`) sample compound model.

```
# Resource Bundle Names
calico.resourcebundle.name=NAResourceBundle
calico.verify.resourcebundle.name=NAVerifyResourceBundle
```

These properties define the resource bundles for the sample pages and the error messages resulting from exceptions thrown during verification.

```
# Default Date Format
```

```
calico.default.date.format=MMM d, yyyy 'at' hh:mm aaa
```

This property defines the date format used by deltas in the sample pages.

```
# Database
calico.na.db=true
```

This property is reserved for functions to be added in a future release. Leave it at its default value.

```
# Images
calico.na.image.violation=/calico/images/violation.gif
calico.na.image.noviolation=/calico/images/no_violation.gif
calico.na.image.add=/calico/images/add_icon.gifcalico.na.image.delete=/calico⇒
/images/delete_icon.gif
calico.na.image.sort=/calico/images/sort_icon.gif
```

These properties specify the images used by the compound configuration sample pages.

```
# Edit Component Entry Points (Relative To Document Root)
Hub=/solutions/CompoundSample/hub/hub.jsp
Node=/solutions/CompoundSample/node/node.jsp
Circuit=/solutions/CompoundSample/circuit/circuit.jsp
```

These properties define the entry points to each model used in compound configuration. Note that all reference the root directory of the web server, and are not relative to the compound configuration manager page or directory.

```
# Network Advisor Application Pages (Relative To Document Root)
calico.page.manager=/solutions/CompoundSample/CalicoNAManager.jsp
```

This property defines the manager page to be used in compound configuration.

```
CalicoNAAddComp=/calico/CalicoNAAddComp.jsp
CalicoNAAddConn=/calico/CalicoNAAddConn.jsp
CalicoNAChangeId=/calico/CalicoNAIdentification.jsp
CalicoNADeleteComp=/calico/CalicoNADeleteComp.jsp
CalicoNADeleteConfig=/calico/CalicoNADeleteConfig.jsp
CalicoNADeleteConn=/calico/CalicoNADeleteConn.jsp
CalicoNAEditComp=/calico/CalicoNAEditComp.jsp
CalicoNAEditConn=/calico/CalicoNAEditConn.jsp
CalicoNAError=/calico/CalicoNAError.jsp
CalicoNALoadConfig=/calico/CalicoNALoadConfig.jsp
CalicoNASaveConfig=/calico/CalicoNASaveConfig.jsp
CalicoNAVerify=/calico/CalicoNAVerify.jsp
CalicoNAVViewBOM=/calico/CalicoNABOM.jsp
CalicoNAVViewDeltaDetail=/calico/CalicoNAVViewDeltaDetail.jsp
CalicoNAVViewDeltas=/calico/CalicoNAVViewDeltas.jsp
```

These properties define the page to be used with each action in compound configuration. When the Manager page finds any of these actions in the request, it will redirect the request to the appropriate page.

File Text

The following is the unannotated text of the CalicoNA.properties file.

```
# PeopleSoft Configurator Compound Modeling Properties
```

```
# Auto Save On Session Timeout (true/false)
calico.session.autosave=false

# Solution info
calico.solution.name=TelcoDemo
calico.solution.version=1-2
calico.solution.description=Compound Modeling Demo
calico.solution.allowNew=true
calico.page.restore=

# Compound Structure File
calico.compoundstructure.name=TelcoDemo
calico.compoundstructure.version=1-0
calico.compoundstructure.restore.policy=1

# Resource Bundle Names
calico.resourcebundle.name=NAResourceBundle
calico.verify.resourcebundle.name=NAVerifyResourceBundle

# Default Date Format
calico.default.date.format=MMM d, yyyy 'at' hh:mm aa

# Database
calico.na.db=true

# Images
calico.na.image.violation=/calico/images/violation.gif
calico.na.image.noviolation=/calico/images/no_violation.gif
calico.na.image.add=/calico/images/add_icon.gif
calico.na.image.delete=/calico/images/delete_icon.gif
calico.na.image.sort=/calico/images/sort_icon.gif

# Edit Component Entry Points (Relative To Document Root)
Hub=/solutions/CompoundSample/hub/hub.jsp
Node=/solutions/CompoundSample/node/node.jsp
Circuit=/solutions/CompoundSample/circuit/circuit.jsp

# Network Advisor Application Pages (Relative To Document Root)
calico.page.manager=/solutions/CompoundSample/CalicoNAManager.jsp

CalicoNAAddComp=/calico/CalicoNAAddComp.jsp
CalicoNAAddConn=/calico/CalicoNAAddConn.jsp
CalicoNAChangeId=/calico/CalicoNAIdentification.jsp
CalicoNADeleteComp=/calico/CalicoNADeleteComp.jsp
CalicoNADeleteConfig=/calico/CalicoNADeleteConfig.jsp
CalicoNADeleteConn=/calico/CalicoNADeleteConn.jsp
CalicoNAEditComp=/calico/CalicoNAEditComp.jsp
CalicoNAEditConn=/calico/CalicoNAEditConn.jsp
CalicoNAError=/calico/CalicoNAError.jsp
```

```
CalicoNALoadConfig=/calico/CalicoNALoadConfig.jsp  
CalicoNASaveConfig=/calico/CalicoNASaveConfig.jsp  
CalicoNAVerify=/calico/CalicoNAVerify.jsp  
CalicoNAViewBOM=/calico/CalicoNABOM.jsp  
CalicoNAViewDeltaDetail=/calico/CalicoNAViewDeltaDetail.jsp  
CalicoNAViewDeltas=/calico/CalicoNAViewDeltas.jsp
```

APPENDIX E

Node-Circuit-Hub Service

This appendix describes a sample complex product (communications services) offering that includes node, circuits, and hubs, and gives you the XML representation of its compound structure definition.

Description of Services

The following bulleted items describe the sample product offering.

Node

In the sample model, a Node:

- Can be created.
- Is configured.
- Can stand alone.
- Can be connected to a single Circuit.

Hub

In the sample model, a Hub:

- Can be created.
- Is configured.
- Can stand alone.
- Can be connected to, at most, 24 Circuits.

Circuit

In the sample model, a Circuit:

- Can be created.
- Is configured.

Connections

In the sample model, applicable conditions for a Connection are:

- Must be connected to at least one Node or Hub.
- Can have at most two connections.
- The one or two connections can be to either Nodes or Hubs.

- Node-Circuit
- Hub-Circuit
- Node-Circuit-Node
- Node-Circuit-Hub or Hub-Circuit-Node
- Hub-Circuit-Hub
- When a Circuit is connected to either a Node or a Hub, the following data relationship exists:
 - The PortSpeedSelection from the first Node or Hub should be applied to the Ext_PortSpeedSelection_A on the Circuit.
 - The PortSpeedSelection from the second Node or Hub should be applied to the Ext_PortSpeedSelection_B on the Circuit.
 - The ProtocolSelection from the first Node or Hub should be applied to the Ext_ProtocolSelection_A on the Circuit.
 - The ProtocolSelection from the second Node or Hub should be applied to the Ext_ProtocolSelection_B on the Circuit.
- When Circuits are connected to a Hub, a collection containing the portSpeed value from each Circuit should be applied to the Ext_PortSpeeds variable on the Hub.

XML Representation of Compound Structure Definition

The following XML represents the compound structure definition for the sample services offering. Compound models use the XML schema at run time to control operation, and to verify the structure of a compound configuration instance. It is located in

```

\bea\weblogic81\config\CalicoDomain\applications\CalicoApp\
solutions\CompoundSample
<CompoundStructure name="TelcoDemo" version="1-0">
  <comment>This is a sample compound structure that represents the current=>
Network Advisor 1.0 TelcoDemo, plus additional functionality available with=>
Network Advisor 3.5. Specifically, this sample adds in a hub component.</comment>
  <Components>
    <ConfigurableComponent name="Node">
      <Model name="TelcoComp" version="1-1"/>
    </ConfigurableComponent>
    <ConfigurableComponent name="Hub">
      <Model name="TelcoCompHub" version="1-1"/>
    </ConfigurableComponent>
    <ConfigurableComponent name="Circuit">
      <Model name="TelcoCompCircuit" version="1-1"/>
      <RequiredRelationships>
        <Relationship ref="Connection"/>
      </RequiredRelationships>
    </ConfigurableComponent>
  </Components>
</Relationships>

```

```

<Relationship name="Connection" component="Circuit">
  <Structure minOccurs="1" maxOccurs="2">
    <ConnectedComponent ref="Node" minOccurs="0" maxOccurs="2"/>
    <ConnectedComponent ref="Hub" minOccurs="0" maxOccurs="2"/>
  </Structure>
  <ConnectionPoints>
    <ConnectionPoint name="PortSpeed" operation="CHOICE" sourceDP="Port⇒
SpeedSelection">
      <comment>Apply the port speed selection</comment>
      <ConnectedComponent instance="1" targetDP="Ext_PortSpeedSelection_⇒
A"/>
      <ConnectedComponent instance="2" targetDP="Ext_PortSpeedSelection_⇒
B"/>
    </ConnectionPoint>
    <ConnectionPoint name="Protocol" operation="CHOICE" sourceDP="Protocol⇒
Selection">
      <comment>Apply the protocol selection</comment>
      <ConnectedComponent instance="1" targetDP="Ext_ProtocolSelection_A"⇒
/>
      <ConnectedComponent instance="2" targetDP="Ext_ProtocolSelection_B"⇒
/>
    </ConnectionPoint>
  </ConnectionPoints>
</Relationship>
<Relationship name="HubCircuits" component="Hub">
  <Structure>
    <ConnectedComponent ref="Circuit" minOccurs="1" maxOccurs="24"/>
  </Structure>
  <ConnectionPoints>
    <ConnectionPoint name="PortSpeeds" operation="COLLECTION" target⇒
Variable="Ext_PortSpeeds">
      <comment>Apply the port speeds from all circuits</comment>
      <ConnectedComponent instance="all" sourceNumericData="portSpeed"/>
    </ConnectionPoint>
  </ConnectionPoints>
</Relationship>
<Relationship name="NodeCircuit" component="Node">
  <Structure>
    <ConnectedComponent ref="Circuit" minOccurs="1" maxOccurs="1"/>
  </Structure>
</Relationship>
</Relationships>
</CompoundStructure>

```


APPENDIX F

Mobile Configurator COM API

This appendix lists the classes and methods of and PeopleSoft.MCfgProxy.

PeopleSoft.MCfgProxy Methods

PeopleSoft.MCfgProxy contains these methods.

Start

Initiates the startup sequence of the Mobile Configurator application. Call this method right after MCfg object creation and before other methods are called on Mobile Configurator. By default the MCfg application window is shown centered and normal.

Arguments

None

Return Values

Type: Integer

0 – Successful

1 – Fail

Stop

Closes and shuts down the Mobile Configurator.

Arguments

None

Return Values

Type: Integer

0 – Successful

1 – Fail

Show

Shows/Restores the Mobile Configurator application window if it was previously hidden on a Hide call.

Arguments

None

Return Values

Type: Integer

0 – Successful

1 – Fail

Hide

Hides the Mobile Configurator application.

Arguments

None

Return Values

Type: Integer

0 – Successful

1 – Fail

NewConfiguration(String sSolutionID)

Loads/starts a new configuration session based on the passed configuration ID.

Arguments

sSolutionID – a string value for the solution ID to load, i.e. “Sample”, “TelcoHub” etc.

Return Values

Type: Integer

0 – Success

1 – Fail

ReConfigure(String sXML)

Reloads/restarts a configuration session based on the previously saved configuration xml formatted record that is passed in.

Arguments

sXML – a string containing the xml formatted configuration record that is to be reloaded for reconfiguration.

Return Values

Type: Integer

0 – Success

1– Fail

WaitForConfiguration(Integer nTimeout)

Gets the state of the configuration session or how it ended (timeout). In doing so, the method waits for the configuration session to end before returning the exit state of the session.

Arguments

nTimeout—length of time in seconds.

Return Values

Type: Integer

0 – Success

1 – Fail

2 – Timeout

3 – Cancel

5 – Configuration Timer Timeout

IsConfigurationReady

Gets the state of the configuration session. In contrast to WaitForConfiguration, this method does not wait for the configuration session to end to send the state but returns it immediately.

Arguments

None

Return Values

Type: Integer

0 – Success

1 –Fail

3 –Cancel

4 – In Progress

5 – Configuration Timer Timeout

GetConfiguration

Retrieves the xml formatted configuration of the configuration session.

Arguments

None

Return Values

Type: String

The xml formatted string of the configuration.

SetLabelToString(Integer nLabelID, String NewLabel)

Sets the specified label, which corresponds to the ID passed in, to the new label string passed in. The label is for the “Save” and “Cancel” menu items.

Arguments

NLabelID – ID of label whose string value is to be set .

Valid values are as follows:

0 – label for “Save”

1 – label for “Cancel”

NewLabel – string value that the label is to be set to.

Return Values

Type: Integer

0 – Success

1 – Fail

StartCountdown(Integer nCountdownTimeInMins, Integer nWarnLeftTimeInMins, String strWarnMessage, String strContinueLabel, String strStopLabel)

Initiates the countdown timers that set the maximum time duration for a configuration session.

Arguments

nCountdownTimeInMins—total length of time in minutes that the configuration can be worked on.

nWarnLeftTimeInMins—the time remaining in minutes that is left in the nCountdownTimeInMins argument, before a message appears to warn the user of the impending timeout, and to ask the user for a course of action (options are to Stop waiting for Mobile Configurator to return or to Continue to wait).

strWarnMessage—the message is displayed in the warning message dialog.

strContinueLabel—the string label for the button that is associated with the user choosing to continue with the configuration session.

StrStoplabel—the string label for the button that is associated with the user choosing to stop the configuration session and save it back to the order.

Return Values

Type: Integer

0 – Success

1 – Fail

StopCountdown

Stops the countdown timers that were previously initiated with the startcountdown method.

Arguments

None

Return Values

Type: Integer

0 – Success

1 – Fail

Sample Application Client Code

The following is sample code included with the application.

```

Dim oMCfg As Object
Set oMCfg = CreateObject("PeopleSoft.MCfGProxy")

If (oMCfg.Start() = 1) Then
    MsgBox "Failed creating the configurator app"
    Set oMCfg = Nothing
End
End If

oMCfg.Show

oMCfg.StartCountdown(30, 10, "Config will timeout in 10
minutes!!", =>
"Continue with session", "Save and return to order")

oMCfg.NewConfiguration txtConfigID.Text

Dim ret As Long

'timeout in 60 seconds
ret = oMCfg.WaitForConfiguration(60)

If (ret = 0) Then
    txtXML.Text = oMCfg.GetConfiguration()
ElseIf (ret = 1) Then
    txtXML.Text = "Configuration failed"
ElseIf (ret = 2) Then
    txtXML.Text = "Configuration timeout"
ElseIf (ret = 3) Then
    txtXML.Text = "Configuration canceled by user"

```

```
ElseIf (ret = 5) Then
    txtXML.Text = "Configuration timer timeout"
End If

oMCfg.StopCountdown

oMCfg.Hide

oMCfg.Stop
Set oMCfg = Nothing
```

APPENDIX G

PCIF

The PeopleSoft Advanced Configurator Interchange Format, or PCIF, is an XML format that can be used to generate PeopleSoft Visual Modeler models from an outside data source. PCIF encapsulates all the functionality that is available to a Configurator modeler.

This appendix describes the elements and ordering of the PCIF document so that you can create an XML file that describes your model data in the format understood by the Configurator Visual Modeler.

Included with the installation of PeopleSoft Enterprise Visual Modeler is the file PCIF.dtd, which describes the structure of PCIF documents and acts as the validator for them.

In order to successfully describe and validate the outside data, the contents of the PCIF document must be ordered so that the Visual Modeler can recognize the elements. To some extent, the DTD enforces this ordering; however, for some objects, particularly CLASS elements, the PCIF writer must maintain a hierarchical ordering to ensure that the document can be imported correctly. Order is explained in greater detail in the individual element sections below.

The information in this appendix assumes that you are familiar with the concepts inherent to the PeopleSoft Configurator, the Visual Modeler, and XML. Some elements' descriptions reference a "target model." The target model is the Visual Modeler model into which PCIF is being imported. The target model does not have to be empty; in fact, a modeler can use a PCIF document to update models with newer data, as the Visual Modeler gives the modeler the option of overwriting objects that have the same name as those defined in an imported PCIF document.

This document is arranged in the order that one might find XML elements in a PCIF document, starting with the topmost element (a MODEL).

Note. It is strongly recommended that any PCIF document created by an application other than the Visual Modeler has the document type declaration:

```
<!DOCTYPE MODEL PUBLIC "PeopleSoft Configurator Model" "PCIF.dtd">
```

This document type declaration should be made before the root element MODEL. Using this document type declaration ensures that when the PCIF document is imported into the Visual Modeler, it will be validated against the DTD, thereby preventing errors in the model due to badly formed PCIF.

MODEL Element

The MODEL element is the root element of a model in the PCIF representation.

Attributes:

Name	Type	Required?	Description
VERSION	CDATA	Yes	The version of PCIF implemented. This document describes PCIF version 1.0.
NAME	CDATA	Yes	The name of the model. On import, this attribute will be overridden by the name of the model importing it.
MODEL_VERSION	CDATA	Yes	The version of the model itself (as would be specified in the Visual Modeler's Project Settings dialog box). On import, this attribute will be overridden by the version of the model importing it.

Subelements:

Name	How Many?	Description
DATABASE_REFERENCE	0 or more	A reference to an external data source.
CLASS	0 or more	A class. Classes must be defined in hierarchical order, i.e. a parent class must be defined before its subclasses. If any classes have SQL queries, their respective DATABASE_REFERENCES should have already been defined.
SELECTION_POINT	0 or more	Selection points must be defined on classes that exist in the target model or have been previously defined in this PCIF document. If a leaf class does not have a selection point defined, the Visual Modeler optionally can autogenerate a selection point for that class.
EXPRESSION	0 or more	An expression.
NOT_COMPATIBLE	0 or more	A non-compatibility constraint.
COMPATIBLE	0 or more	A compatibility constraint.
REQUIRED	0 or more	A requires constraint.
DYNAMIC_DEFAULT	0 or more	A dynamic default.

Name	How Many?	Description
ELIMINATION	0 or more	An elimination constraint.
COMPARISON	0 or more	A comparison constraint.
RESOURCE_CONSTRAINT	0 or more	A resource balancing constraint.
SUMMATION	0 or more	A summation.

DATABASE_REFERENCE Element

Attributes:

Name	Type	Required?	Description
ALIAS	CDATA	Yes	The name by which this database will be referenced in the Visual Modeler.
DATA_SOURCE	CDATA	Yes	The name of the underlying ODBC and JDBC data source.
LOGIN_ID	CDATA	Yes	The login name for the data source.
PASSWORD	CDATA	Yes	The password for the data source. If the Visual Modeler has created the PCIF file, this password will be encrypted using the Visual Modeler's password encryption scheme. For security reasons, an encrypted password can only be read and written by the Visual Modeler.
PASSWORD_ENCRYPTED	Boolean	Yes	A flag that tells the Visual Modeler if the password specified in the PASSWORD attribute has been encrypted. If an application other than the Visual Modeler is creating a PCIF document, this attribute should always be set to FALSE, and the password should be unencrypted.

Subelements: None.

CLASS Element

Classes must be defined in a hierarchical manner; that is, parent classes must be defined in a MODEL before the child classes are defined. Furthermore, attributes that are referenced in a class' domain members or SQL query should be defined on the class or on one of its parent classes.

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the class. Each individual class must have a unique name; if a duplicate name is found in the model, and the class of that name has the same parent class, then the Visual Modeler can optionally overwrite the existing class with the class defined in the PCIF. If the class's name in PCIF is "RootClass", any attributes defined on this class will be added to the Root Class.
PARENT	CDATA	Yes	The parent class of this class. If not specified, the class will be a child of the Root Class by default.

Subelements:

Name	How Many?	Description
CLASS_ATTRIBUTE	0 or more	An attribute defined on this class. There should never be attribute conflicts with a parent class; that is, if an attribute is defined on a parent class, no attribute by that name should be defined on any child class.
DOMAIN_MEMBER	0 or more	A domain member. Classes should not have both a SQL query and domain members; the two are mutually exclusive. Classes that are parent classes (i.e. they have subclasses) should not have domain members.

Name	How Many?	Description
STANDARD_QUERY	0 or 1	A standard SQL query that defines an database from which this class should get its domain members. If a STANDARD_QUERY is defined on a CLASS, that CLASS should not have any DOMAIN_MEMBER elements, and it should not have an ADVANCED_QUERY element.
ADVANCED_QUERY	0 or 1	An advanced SQL query that defines an database from which this class should get its domain members. If an ADVANCED_QUERY is defined on a CLASS, that CLASS should not have any DOMAIN_MEMBER elements, and it should not have an STANDARD_QUERY element.
REFRESH_INTERVAL	0 or more	A rule that defines how often this class will refresh those attributes which have been marked for scheduled data refreshes. Should only be present if STANDARD_QUERY or ADVANCED_QUERY is defined. Multiple REFRESH_INTERVALs will effectively be “anded” together, so all of them will apply.
SELECTION_POINT_ATTRIBUTE	0 or more	A selection point level attribute defined on this class. There should never be attribute conflicts with a parent class or with selection points defined on this class or its subclasses; that is, if a selection point level attribute is defined here, a selection point level attribute of that name should not be defined on the parent class, subclasses, or selection points of this class. Furthermore, no domain member level attributes of that name should be defined on the parent class or subclasses of this class.

CLASS_ATTRIBUTE Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the attribute. This attribute name should not have been defined on an parent of the class it's currently being defined on.
TYPE	Choice of {Date, String, Boolean, Int, Float}	Yes	The type of this attribute.

Subelements:

Name	How Many?	Description
DEFAULT_VALUE	0 or 1	The default value of this attribute.

DEFAULT_VALUE Element

Attributes: None.

Subelements: None.

The DEFAULT_VALUE element contains #PCDATA , which holds the default value of a CLASS_ATTRIBUTE. If the CLASS_ATTRIBUTE is of type Date, the default value should be formatted as an ISO-standard time, such as 2002-09-17T00:00:00.000000+06:00.

If the CLASS_ATTRIBUTE is of type Float, the default value should be formatted as a floating point number, such as 1.328.

If the CLASS_ATTRIBUTE is of type Integer, the default value should be formatted as an integer number, such as 13.

If the CLASS_ATTRIBUTE is of type Boolean, the default value should be formatted as a boolean with the first letter capitalized, such as True or False.

DOMAIN_MEMBER Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the domain member. If this domain member already exists in the target model, properties from the PCIF domain member will overwrite those in the target model.

Subelements:

Name	How Many?	Description
DEFAULT_VALUE	0 or 1	The default value of this attribute.

DM_ATTRIBUTE Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the attribute for which this domain member is providing a value.

Subelements:

None.

The DM_ATTRIBUTE element contains #PCDATA, which holds a value. If the CLASS_ATTRIBUTE to which this DM_ATTRIBUTE is referring is of type Date, the default value should be formatted as an ISO-standard time, such as 2002-09-17T00:00:00.000000+06:00.

If the CLASS_ATTRIBUTE is of type Float, the default value should be formatted as a floating point number, such as 1.328.

If the CLASS_ATTRIBUTE is of type Integer, the default value should be formatted as an integer number, such as 13.

If the CLASS_ATTRIBUTE is of type Boolean, the default value should be formatted as a boolean with the first letter capitalized, such as True or False.

STANDARD_QUERY Element

Attributes:

Name	Type	Required?	Description
DATA_SOURCE	CDATA	Yes	A reference to the ALIAS of a DATABASE_REFERENCE.
DISTINCT	Boolean	Yes	Flag that decides whether the query retrieves distinct results.

Subelements:

Name	How Many?	Description
PRIMARY_TABLE	1	The primary WHERE clause and domain member mappings.
SECONDARY_TABLE	0 or more	JOIN clauses and secondary domain member mappings.

PRIMARY_TABLE Element

Attributes:

Name	Type	Required?	Description
TABLE	CDATA	Yes	The table in the database where this query will be retrieving its data.

Subelements:

Name	How Many?	Description
COLUMN	0 or more	The mappings of database table columns to domain member attributes.
WHERE	0 or 1	The query's where clause. An empty clause will select all the specified columns of the table unconditionally.

COLUMN Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the column coming from the database table.
KEY	Boolean	Yes	Flag that decides whether this column will be a domain member key. Exactly one column should have this attribute set to TRUE.
ATTRIBUTE_NAME	CDATA	Yes	The domain member attribute into which this column's data will be stored.

Subelements: None

WHERE Element

Attributes: None

Subelements: None.

Contains #PCDATA which acts as the WHERE clause for a standard SQL query.

SECONDARY_TABLE Element

Attributes:

Name	Type	Required?	Description
TABLE	CDATA	Yes	The table in the database where this query will be retrieving its data.

Subelements:

Name	How Many?	Description
COLUMN	0 or more	The mappings of database table columns to domain member attributes.
JOIN	0 or 1	The query's join clause. An empty clause will join all the specified columns of the table unconditionally.

JOIN Element

Attributes: None.

Subelements: None.

Contains #PCDATA which acts as the JOIN clause for a standard SQL query.

ADVANCED_QUERY Element

Attributes:

Name	Type	Required?	Description
DATA_SOURCE	CDATA	Yes	A reference to the ALIAS of a DATABASE_REFERENCE.
ASSUME_SORTED	Boolean	No	Flag that indicates whether the ViM should assume the data from this query is already sorted in the order the modeler requires

Subelements:

Name	How Many?	Description
COLUMN	0 or more	The mappings of database table columns to domain member attributes.
QUERY_TEXT	0 or 1	The text of the SQL query.

QUERY_TEXT Element

Attributes: None.

Subelements: None.

Contains #PCDATA, which acts as the text of an SQL query.

SELECTION_POINT Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the selection point. Each individual selection point must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing selection point with the selection point defined in the PCIF.
CLASS	CDATA	Yes	The name of the class this selection point refers to. This class must have been defined either in the target model or earlier in the PCIF document.

Name	Type	Required?	Description
DEFAULT_QUANTITY	CDATA	No	The default quantity to use for all domain members in this selection point. Cannot be specified if DEFAULT_QUANTITY_EXPRESSION is specified.
DEFAULT_QUANTITY_POLICY	CDATA	No	The quantity policy to use with defaults applied to all domain members in the selection point. Only valid if DEFAULT_QUANTITY or DEFAULT_QUANTITY_EXPRESSION is specified.
DEFAULT_QUANTITY_EXPRESSION	CDATA	No	The expression that defines the default quantity to use for all domain members in this selection point. Cannot be specified if DEFAULT_QUANTITY is specified.
QUANTITY	Boolean	No	Flag that defines whether this selection point has quantity. Defaults to false.
QUANTITY_ATTRIBUTE	CDATA	No	Domain member attribute from which to gather default quantity data.
QUANTITY_ATTRIBUTE_POLICY	CDATA	No	Quantity policy to use with defaults gathered from domain member attributes. Only valid when QUANTITY_ATTRIBUTE is specified.
MULTISELECT	Boolean	No	Flag that defines whether this selection point is multiselect. Defaults to false.
OPTIONAL	Boolean	No	Flag that defines whether this selection point is optional. Defaults to false.
USE_MIN_MAX	Boolean	No	Flag that defines whether this selection point uses the min and max quantity constraints. Defaults to false.

Subelements:

Name	How Many?	Description
STATIC_DEFAULTS	0 or 1	The set of static defaults for this selection point.
SELECTION_POINT_MIN_QTY_SETTINGS	0 or 1	The minimum quantity settings for the selection point. This subelement should only be defined if USE_MIN_MAX is true.
SELECTION_POINT_MAX_QTY_SETTINGS	0 or 1	The maximum quantity settings for the selection point. This subelement should only be defined if USE_MIN_MAX is true.
DOMAIN_MEMBER_MIN_QTY_SETTINGS	0 or 1	The minimum quantity settings for domain members on this selection point. This subelement should only be defined if both USE_MIN_MAX and QUANTITY are true.
DOMAIN_MEMBER_MAX_QTY_SETTINGS	0 or 1	The maximum quantity settings for domain members on this selection point. This subelement should only be defined if both USE_MIN_MAX and QUANTITY are true.
SELECTION_POINT_ATTRIBUTE	0 or more	A selection point level attribute defined on this selection point. There should never be attribute conflicts with the class on which this selection point is defined; that is, if a selection point level attribute is defined here, a selection point level attribute of that name should not be defined this selection point's class or any of its parent classes. Furthermore, no domain member level attributes of that name should be defined on this selection point's class or any of its parent classes.

STATIC_DEFAULTS Element

Attributes: None

Subelements:

Name	How Many?	Description
STATIC_DEFAULT	0 or more	A static default.

STATIC_DEFAULT Element

Attributes:

Name	Type	Required?	Description
DOMAIN_MEMBER	CDATA	Yes	The domain member that is to be defaulted.
QUANTITY	CDATA	No	The quantity that should be used with this default. Should be an integer number. Should not be defined if QUANTITY_EXPRESSION is defined.
QUANTITY_EXPRESSION	CDATA	No	The expression that should be used to find the quantity for this default. Should not be defined if QUANTITY is defined.
QUANTITY_POLICY	CDATA	No	The quantity policy that should be used with this default. Should be one of {IGNORE, MIN, MAX, SUM}. IGNORE corresponds to a quantity policy of "Overridable."

Subelements: None

SELECTION_POINT_MIN_QTY_SETTINGS Element

Attributes:

Name	Type	Required?	Description
SOURCE	Choice of {QUANTITY, EXPRESSION, SQL_QUERY}	Yes	The source of the number that will be the minimum number of selections on the selection point.
QUANTITY	CDATA	No	The absolute quantity that will act as the minimum selections. Should only be specified if SOURCE is QUANTITY. Should be an integer.
EXPRESSION	CDATA	No	The expression that will act as the minimum selections. Should only be specified if SOURCE is EXPRESSION.

Name	Type	Required?	Description
SQL_QUERY	CDATA	No	The SQL query that will gather the data to act as the minimum selections. Should only be specified if SOURCE is SQL_QUERY.
DATABASE	CDATA	No	The ALIAS of the database that the SQL query will act on. Should be defined in the target model or in the PCIF earlier as a DATABASE_REFERENCE. Should only be specified if SOURCE is SQL_QUERY.

Subelements:

Name	How Many?	Description
EXPLANATION	0 or 1	The explanation that will appear if this constraint is violated.

DOMAIN_MEMBER_MIN_QTY_SETTINGS Element

Attributes:

Name	Type	Required?	Description
SOURCE	Choice of {QUANTITY, EXPRESSION, ATTRIBUTE}	Yes	The source of the number that will be the minimum number of selections for each domain member.
QUANTITY	CDATA	No	The absolute quantity that will act as the minimum quantity. Should only be specified if SOURCE is QUANTITY. Should be an integer.
EXPRESSION	CDATA	No	The expression that will act as the minimum quantity. Should only be specified if SOURCE is EXPRESSION.
ATTRIBUTE	CDATA	No	The attribute that will contain the data to act as the minimum quantity. Should only be specified if SOURCE is ATTRIBUTE. The attribute should already have been defined on the selection point's class or one of its parent classes.

Subelements:

Name	How Many?	Description
EXPLANATION	0 or 1	The explanation that will appear if this constraint is violated.

DOMAIN_MEMBER_MAX_QTY_SETTINGS Element

Attributes:

Name	Type	Required?	Description
SOURCE	Choice of {QUANTITY, EXPRESSION, ATTRIBUTE}	Yes	The source of the number that will be the maximum number of selections for each domain member.
QUANTITY	CDATA	No	The absolute quantity that will act as the maximum quantity. Should only be specified if SOURCE is QUANTITY. Should be either an integer or the word "UNBOUNDED".
EXPRESSION	CDATA	No	The expression that will act as the maximum quantity. Should only be specified if SOURCE is EXPRESSION.
ATTRIBUTE	CDATA	No	The attribute that will contain the data to act as the maximum quantity. Should only be specified if SOURCE is ATTRIBUTE. The attribute should already have been defined on the selection point's class or one of its parent classes.

Subelements:

Name	How Many?	Description
EXPLANATION	0 or 1	The explanation that will appear if this constraint is violated.

DYNAMIC_DEFAULT Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the dynamic default. Each individual dynamic default must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing dynamic default with the dynamic default defined in the PCIF. If a constraint of a different type exists in the model, the ViM will not import this constraint.
ENABLED	Boolean	No	Flag that specifies whether the constraint is enabled. Defaults to TRUE.
FORMAT	Choice of {Internal, DBTable, SQLQuery}	No	Specifies whether this constraint stores its row data internally (to CMS files), stores them to a database table, or retrieves them from a SQL Query.
DATA_SOURCE	CDATA	No	Specifies from which database this constraint should store or retrieve its data. Must be a reference to a database name previously defined in the PCIF file as a DATABASE_REFERENCE, or to a database already defined in the destination model. The DATA_SOURCE attribute must be present if FORMAT is DBTable or SQLQuery.

Subelements:

Name	How Many?	Description
EFFECTIVITY	0 or 1	The effectivity dates for this constraint.
COMMENT	0 or 1	A comment on this constraint.
EXPLANATION	0 or 1	An explanation that is shown if this constraint is violated.
ARGUMENT	1 or more	An argument that displays as a column on the left hand side.
RHS_ARGUMENT	1 or more	An argument that displays as a column on the right hand side.
ROW	1 or more	A row that defines a set of values that are incompatible.
CONSTRAINT_QUERY	0 or 1	The SQL query information for this constraint. Should only be specified if the FORMAT attribute is set to SQLQuery.

EXPLANATION Element

Attributes: None.

Subelements: None.

Contains #PCDATA with the text of an explanation.

EXPRESSION Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the expression. Each individual expression must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing expression with the expression defined in the PCIF.
TYPE	Choice of {Numeric, Boolean, Date, String}	Yes	The return type of the expression.
COMMENT	CDATA	No	A comment on the expression.

Subelements: None.

Contains #PCDATA which represents the text of the expression itself.

NOT_COMPATIBLE Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the non-compatibility constraint. Each individual non-compatibility must have a unique name; if a duplicate name is found in the model, then the ViM can optionally overwrite the existing non-compatibility with the non-compatibility defined in the PCIF. If a constraint of a different type exists in the model, the ViM will not import this constraint.

Name	Type	Required?	Description
ENABLED	Boolean	No	Flag that specifies whether the constraint is enabled. Defaults to TRUE.
FORMAT	Choice of {Internal, DBTable, SQLQuery}	No	Specifies whether this constraint stores its row data internally (to CMS files), stores them to a database table, or retrieves them from a SQL Query.
DATA_SOURCE	CDATA	No	Specifies from which database this constraint should store or retrieve its data. Must be a reference to a database name previously defined in the PCIF file as a DATABASE_REFERENCE, or to a database already defined in the destination model. The DATA_SOURCE attribute must be present if FORMAT is DBTable or SQLQuery.

Subelements:

Name	How Many?	Description
EFFECTIVITY	0 or 1	The effectivity dates for this constraint.
COMMENT	0 or 1	A comment on this constraint.
EXPLANATION	0 or 1	An explanation that is shown if this constraint is violated.
ARGUMENT	1 or more	An argument that displays as a column on the left hand side.
RHS_ARGUMENT	0 or more	An argument that displays as a column on the right hand side.
ROW	0 or more	A row that defines a set of values that are incompatible.
CONSTRAINT_QUERY	0 or 1	The SQL query information for this constraint. Should only be specified if the FORMAT attribute is set to SQLQuery.

EXTERN Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name by which this extern will be referenced in the Visual Modeler.
DEFAULT_VALUE	CDATA	No	The default value of this extern.
REQUIRED	Boolean	No	Flag indicating whether a value must be provided for this extern at runtime in order for the configuration to be valid.
TYPE	Choice of {Date, String, Boolean, Int, Float}	Yes	The data type of this extern.

Subelements: None

EFFECTIVITY Element

Attributes: None

Subelements:

Name	How Many?	Description
EFFECTIVEDATE	1 or more	The effectivity dates.

EFFECTIVEDATE Element

Attributes:

Name	Type	Required?	Description
EFFECTIVE_FROM	CDATA	Yes	The date effectivity begins. Should be in ISO-time format, e.g. 2002-09-17T00:00:00.000000+06:00. Should be a date before EFFECTIVE_TO.
EFFECTIVE_TO	CDATA	Yes	The date effectivity ends. Should be in ISO-time format, e.g. 2002-09-17T00:00:00.000000+06:00. Should be a date after EFFECTIVE_FROM.
COMMENT	CDATA	No	A comment on these effective dates.

Subelements:

Name	How Many?	Description
EFFECTIVEDATE	1 or more	The effectivity dates.

COMMENT Element

Attributes: None.

Subelements: None.

Contains #PCDATA with the text of a comment.

ARGUMENT Element

Attributes:

Name	Type	Required?	Description
LEVEL	CDATA	No	The level of this argument. Should be an integer. Defaults to 1.
CLASS	CDATA	No	The class this argument refers to. Either the CLASS or the SELECTION_POINT must be specified, but not both.
SELECTION_POINT	CDATA	No	The selection point this argument refers to. Either the CLASS or the SELECTION_POINT must be specified, but not both.
PROPERTY	CDATA	No	The property (also known as an attribute) that this argument refers to.
EXPRESSION	CDATA	No	The expression that this argument refers to.
COMPLETE	Boolean	No	Flag indicating whether the set of expression comparisons in the column denoted by this argument forms a complete range. Only valid if the EXPRESSION attribute is specified.

Subelements: None

RHS_ARGUMENT Element

Attributes:

Name	Type	Required?	Description
LEVEL	CDATA	No	The level of this argument. Should be an integer. Defaults to 1.
CLASS	CDATA	No	The class this argument refers to. Either the CLASS or the SELECTION_POINT must be specified, but not both.
SELECTION_POINT	CDATA	No	The selection point this argument refers to. Either the CLASS or the SELECTION_POINT must be specified, but not both.
PROPERTY	CDATA	No	The property (also known as an attribute) that this argument refers to.

Subelements: None

ROW Element

Attributes: None

Subelements:

Name	How Many?	Description
VALUE	0 or more	A single value in a column of a table constraint. If this SET is on the left hand side, there should be only one VALUE. If it is on the right hand side, there can be multiple VALUES.

There should be exactly as many SET elements in a ROW as there are ARGUMENTs plus RHS_ARGUMENTs. For instance, if there are two ARGUMENTs and three RHS_ARGUMENTs, there should be five SET elements in each ROW.

SET Element

Attributes: None

Subelements:

Name	How Many?	Description
DEFAULT_VALUE	0 or 1	The default value of this attribute.

VALUE Element

Attributes: None.

Subelements: None.

Contains #PCDATA that references a domain member name or attribute value.

The CONSTRAINT_QUERY Element

Attributes:

Name	Type	Required?	Description
SEPARATOR_CHARACTER	CDATA	No	If this SQL query will return multiple values per row, this is the character that separates each individual value. Should be exactly one character long.
QUANTITY_SEPARATOR	CDATA	No	If this SQL query will return values with quantities and quantity policies, this is the character that will separate the policy, the quantity, and the value. Should be exactly one character long.

Subelements:

Name	How Many?	Description
SQL_CLAUSE	Exactly 1	The SQL clause for this constraint.

SQL_CLAUSE Element

Attributes: None.

Subelements: None.

Contains #PCDATA that is a complete SQL clause.

COMPATIBLE Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the compatibility constraint. Each individual compatibility must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing compatibility with the compatibility defined in the PCIF. If a constraint of a different type exists in the model, the Visual Modeler will not import this constraint.
ENABLED	Boolean	No	Flag that specifies whether the constraint is enabled. Defaults to TRUE.
FORMAT	Choice of (Internal, DBTable, SQLQuery)	No	Specifies whether this constraint stores its row data internally (to CMS files) or to a database table, or retrieves them from a SQL Query.
DATA_SOURCE	CDATA	No	Specifies the database that this constraint is to use to store or retrieve its data. Must be a reference to a database name previously defined in the PCIF file as a DATABASE_REFERENCE, or to a database already defined in the destination model. The DATA_SOURCE attribute must be present if FORMAT is DBTable or SQLQuery.

Subelements:

Name	How Many?	Description
EFFECTIVITY	0 or 1	The effectivity dates for this constraint..
COMMENT	0 or 1	A comment on this constraint.
EXPLANATION	0 or 1	An explanation that is shown if this constraint is violated.
ARGUMENT	1 or more	An argument that displays as a column on the left hand side.
RHS_ARGUMENT	0 or more	An argument that displays as a column on the right hand side.
ROW	0 or more	A row that defines a set of values that are incompatible.
CONSTRAINT_QUERY	0 or 1	The SQL query information for this constraint. Should only be specified if the format attribute is set to SQLQuery.

REQUIRED Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the required constraint. Each individual required must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing required with the required defined in the PCIF. If a constraint of a different type exists in the model, the Visual Modeler will not import this constraint.
ENABLED	Boolean	No	Flag that specifies whether the constraint is enabled. Defaults to TRUE.
FORMAT	Choice of (Internal, DBTable, SQLQuery)	No	Specifies whether this constraint stores its row data internally (to CMS files) or to a database table, or retrieves them from a SQL Query.
DATA_SOURCE	CDATA	No	Specifies the database that this constraint is to use to store or retrieve its data. Must be a reference to a database name previously defined in the PCIF file as a DATABASE_REFERENCE, or to a database already defined in the destination model. The DATA_SOURCE attribute must be present if FORMAT is DBTable or SQLQuery.

Subelements:

Name	How Many?	Description
EFFECTIVITY	0 or 1	The effectivity dates for this constraint.
COMMENT	0 or 1	A comment on this constraint.
EXPLANATION	0 or 1	An explanation that is shown if this constraint is violated.
ARGUMENT	1 or more	An argument that displays as a column on the left hand side.
RHS_ARGUMENT	1 or more	An argument that displays as a column on the right hand side.
ROW	0 or more	A row that defines a set of values that are incompatible.
CONSTRAINT_QUERY	0 or 1	The SQL query information for this constraint. Should only be specified if the format attribute is set to SQLQuery.

ELIMINATION Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the elimination constraint. Each individual elimination must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing elimination with the elimination defined in the PCIF. If a constraint of a different type exists in the model, the Visual Modeler will not import this constraint.
ENABLED	Boolean	No	Flag that specifies whether the constraint is enabled. Defaults to TRUE.
LEVEL	CDATA	No	The level of this constraint. Defaults to 1.
ALLOW	Choice of {NONE, ALL, GREATEST, LEAST, FIRST, LAST, EARLIEST, LATEST}	No	The set of values the elimination will allow.
SELECTION_POINT	CDATA	No	The selection point this elimination applies to.
TARGET_ATTRIBUTE	CDATA	No	The attribute this elimination acts upon. Should be a valid attribute of the SELECTION_POINT specified earlier.
COMPARATOR	Choice of {NEQ, GEQ, LEQ, LT, GT, EQ, CONTAINS, DOES_NOT_CONTAIN, STARTS_WITH, ENDS_WITH, STARTS, ENDS, IS_CONTAINED_IN, IS_NOT_CONTAINED_IN}	Yes	The comparator that will be used in the elimination.

Name	Type	Required?	Description
COMPARISON_EXPRESSION	CDATA	No	The expression that the target attribute will be compared to. Should be of the same type as the TARGET_ATTRIBUTE. Should not be specified if COMPARISON_CONSTANT is specified.
COMPARISON_CONSTANT	CDATA	No	The constant that the target attribute will be compared to. Should be of the same type as the TARGET_ATTRIBUTE. Should not be specified if COMPARISON_EXPRESSION is specified.

Subelements:

Name	How Many?	Description
EFFECTIVITY	0 or 1	The name of the comparison constraint. Each individual comparison must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing comparison with the comparison defined in the PCIF. If a constraint of a different type exists in the model, the Visual Modeler will not import this constraint.
COMMENT	0 or 1	A comment on this constraint.
EXPLANATION	0 or 1	An explanation that is shown if this constraint is violated.

COMPARISON Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the comparison constraint. Each individual comparison must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing comparison with the comparison defined in the PCIF. If a constraint of a different type exists in the model, the Visual Modeler will not import this constraint.
ENABLED	Boolean	No	Flag that specifies whether the constraint is enabled. Defaults to TRUE.
LEVEL	CDATA	No	The level of this constraint. Defaults to 1.

Name	Type	Required?	Description
LEFT_EXPRESSION	CDATA	No	The expression on the left hand side of the comparison.
COMPARATOR	Choice of {NEQ, GEQ, LEQ, LT, GT, EQ, CONTAINS, DOES_NOT_CONTAIN, STARTS_WITH, ENDS_WITH, STARTS, ENDS, IS_CONTAINED_IN, IS_NOT_CONTAINED_IN}	No	The comparator that will be used in the elimination.
RIGHT_EXPRESSION	CDATA	No	The expression on the right hand side of the comparison. Should be of the same type as the LEFT_EXPRESSION. Should not be specified if RIGHT_CONSTANT is specified.
RIGHT_CONSTANT	CDATA	No	The constant on the right hand side of the comparison. Should be of the same type as the LEFT_EXPRESSION. Should not be specified if RIGHT_EXPRESSION is specified.

Subelements:

Name	How Many?	Description
EFFECTIVITY	0 or 1	The effectivity dates for this constraint.
COMMENT	0 or 1	A comment on this constraint.
EXPLANATION	0 or 1	An explanation that is shown if this constraint is violated.

RESOURCE_CONSTRAINT Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the resource constraint. Each individual resource constraint must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing resource constraint with the resource constraint defined in the PCIF. If a constraint of a different type exists in the model, the Visual Modeler will not import this constraint.
ENABLED	Boolean	No	Flag that specifies whether the constraint is enabled. Defaults to TRUE.

Subelements:

Name	How Many?	Description
EFFECTIVITY	0 or 1	The effectivity dates for this constraint.
COMMENT	0 or 1	A comment on this constraint.
EXPLANATION	0 or 1	An explanation that is shown if this constraint is violated.
RESOURCE_PROVIDERS	Exactly 1	The resource providers.
RESOURCE_CONSUMERS	Exactly 1	The resource consumers.

RESOURCE_PROVIDERS Element

Attributes: None

Subelements:

Name	How Many?	Description
ATTRIBUTE	0 or more	The attributes that provide resources.

ATTRIBUTE Element

Attributes:

Name	Type	Required?	Description
CLASS	CDATA	Yes	The class this attribute refers to. Either the CLASS or the SELECTION_POINT must be specified, but not both.
SELECTION_POINT	CDATA	No	The selection point this attribute refers to. Either the CLASS or the SELECTION_POINT must be specified, but not both.
PROPERTY	CDATA	Yes	The property (also known as an attribute) that this attribute refers to.

Subelements: None

RESOURCE_CONSUMERS Element

Attributes: None

Subelements:

Name	How Many?	Description
ATTRIBUTE	0 or more	The attributes that consume resources.

SELECTION_POINT_ATTRIBUTE Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the selection point level attribute. This attribute name should not have been defined on a parent of the class or selection point it's currently being defined on.
TYPE	Choice of {Date, String, Boolean, Int, Float}	Yes	The type of this attribute.

Subelements:

Name	How Many?	Description
DEFAULT_VALUE	0 or 1	The default value of this attribute.

SUMMATION Element

Attributes:

Name	Type	Required?	Description
NAME	CDATA	Yes	The name of the summation. Each individual summation must have a unique name; if a duplicate name is found in the model, then the Visual Modeler can optionally overwrite the existing summation with the summation defined in the PCIF. If a constraint of a different type exists in the model, the Visual Modeler will not import this constraint.
ENABLED	Boolean	No	Flag that specifies whether the summation is enabled. Defaults to TRUE.

Subelements:

Name	How Many?	Description
COMMENT	0 or 1	A comment on this constraint.
SUMMANDS	Exactly 1	The attributes to be summed.
TOTAL_ATTRIBUTE	Exactly 1	The attribute where the sum will be stored.

SUMMANDS Element

Attributes: None

Subelements:

Name	How Many?	Description
ATTRIBUTE	0 or more	The attributes that are to be summed.

TOTAL_ATTRIBUTE Element

Attributes:

Name	Type	Required?	Description
CLASS	CDATA	No	The class this attribute refers to. Either the CLASS or the SELECTION_POINT must be specified, but not both.
SELECTION_POINT	CDATA	No	The selection point this attribute refers to. Either the CLASS or the SELECTION_POINT must be specified, but not both.
PROPERTY	CDATA	Yes	The property (also known as an attribute) that this attribute refers to.

Subelements: None

APPENDIX H

Element-Attribute Trees

XML may be viewed as a tree of elements, with some elements having attributes. This appendix presents COP XML as element-attribute trees, and has these sections:

- Complete COP XML
- Configurator XML Interface

Complete COP XML

The COP DTD defines which XML elements and attributes may be used for:

- Configurations saved in XML, and restored from XML by the COP.
- XML requests sent to, and XML responses received from the COP, and translated by the COPXMLServlet—the Configurator XML interface.

The next two trees present the complete COP XML, including those elements and attributes that are not part of the Configurator XML interface.

Without Attributes

This tree presents the complete COP XML without attributes:

```
CONFIGURATION
  DECISION_POINTS
    DP
      ATTR
      DM
      ATTR
    EXPLANATION
  ATTRIBUTE_SET
    ATTRIBUTE
  CHOICES
    CH
    EVCH
  CONTROL_DATA
    ATTR
    DP
      ATTR
      DM
      ATTR
```

```

    EXPLANATION
NUMERIC_VALUES
    NUM
    EXTERN_VARS
    EV
    VAL
VIOLATIONS
    EXPLANATION
    ERROR

```

With Attributes

This tree presents the complete COP XML with attributes.

```

CONFIGURATION MODEL_ID LOCALE MODEL_VERSION COMPILE_VERSION
    TOTAL_PRICE NETWORK_ADVISOR RESTORE_POLICY
DECISION_POINTS ALL
    DP NM CL DPR MS
    ATTR NM
    DM NM CL ST QTY SL EL PR
    ATTR NM
    EXPLANATION
ATTRIBUTE_SET RET
    ATTRIBUTE NAME
CHOICES RET
    CH DP DM BY ST SL EL QTY TY
    EVCH DP VAL TY
CONTROL_DATA DMSORT_QTY DMSORT_ST DMSORT_ATTR SO FILTER_EL_LO
    FILTER_EL_HI FILTER_EL EXPLNATIONS
    ATTR NM
    DP NM CL DPR MS
    ATTR NM
    DM NM CL ST QTY SL EL PR
    ATTR NM
    EXPLANATION
NUMERIC_VALUES
    NUM NM VL
EXTERN_VARS
    EV NM
    VAL
VIOLATIONS EXPLANATIONS
    EXPLANATION
    ERROR
CONFIG_REC RET
ATTRIBUTE_REC RET
CONFIG_XML RET
PRICING_DATA RET
    DP
    ATTR

```

Configurator XML Interface

The Configurator XML interface uses only some of the elements and attributes of the COP DTD. The next two trees present only those elements and attributes that the COPXMLServlet translates into HTTP POST requests, and responses.

Request

This tree has those COP XML elements and attributes that are used in a request.

```

CONFIGURATION MODEL_ID LOCALE MODEL_VERSION COMPILE_VERSION TOTAL_PRICE
DECISION_POINTS ALL
CHOICES RET
  CH DP DM BY QTY
  EVCH DP VAL
CONTROL_DATA DMSORT_ST FILTER_EL_LO FILTER_EL_HI
  FILTER_EL EXPLANATIONS
ATTR NM
DP NM CL DPR
  ATTR NM
NUMERIC_VALUES
EXTERN_VARS
VIOLATIONS EXPLANATIONS

```

Response

This tree has those COP XML elements and attributes that are used in a response.

```

CONFIGURATION MODEL_ID LOCALE MODEL_VERSION COMPILE_VERSION TOTAL_PRICE
DECISION_POINTS
  DP NM
CHOICES
  CH DP DM BY ST SL EL QTY TY
  EVCH DP VAL TY
CONTROL_DATA
  DP NM CL MS
  DM NM CL ST QTY SL EL PR
  ATTR NM
  EXPLANATION
NUMERIC_VALUES
  NUM NM VL
EXTERN_VARS
  EV NM
  VAL
VIOLATIONS EXPLANATIONS
  EXPLANATION
ERROR

```


Glossary of PeopleSoft Enterprise Terms

absence entitlement	This element defines rules for granting paid time off for valid absences, such as sick time, vacation, and maternity leave. An absence entitlement element defines the entitlement amount, frequency, and entitlement period.
absence take	This element defines the conditions that must be met before a payee is entitled to take paid time off.
academic career	In PeopleSoft Enterprise Campus Solutions, all course work that a student undertakes at an academic institution and that is grouped in a single student record. For example, a university that has an undergraduate school, a graduate school, and various professional schools might define several academic careers—an undergraduate career, a graduate career, and separate careers for each professional school (law school, medical school, dental school, and so on).
academic institution	In PeopleSoft Enterprise Campus Solutions, an entity (such as a university or college) that is independent of other similar entities and that has its own set of rules and business processes.
academic organization	In PeopleSoft Enterprise Campus Solutions, an entity that is part of the administrative structure within an academic institution. At the lowest level, an academic organization might be an academic department. At the highest level, an academic organization can represent a division.
academic plan	In PeopleSoft Enterprise Campus Solutions, an area of study—such as a major, minor, or specialization—that exists within an academic program or academic career.
academic program	In PeopleSoft Enterprise Campus Solutions, the entity to which a student applies and is admitted and from which the student graduates.
accounting class	In PeopleSoft Enterprise Performance Management, the accounting class defines how a resource is treated for generally accepted accounting practices. The Inventory class indicates whether a resource becomes part of a balance sheet account, such as inventory or fixed assets, while the Non-inventory class indicates that the resource is treated as an expense of the period during which it occurs.
accounting date	The accounting date indicates when a transaction is recognized, as opposed to the date the transaction actually occurred. The accounting date and transaction date can be the same. The accounting date determines the period in the general ledger to which the transaction is to be posted. You can only select an accounting date that falls within an open period in the ledger to which you are posting. The accounting date for an item is normally the invoice date.
accounting split	The accounting split method indicates how expenses are allocated or divided among one or more sets of accounting ChartFields.
accumulator	You use an accumulator to store cumulative values of defined items as they are processed. You can accumulate a single value over time or multiple values over time. For example, an accumulator could consist of all voluntary deductions, or all company deductions, enabling you to accumulate amounts. It allows total flexibility for time periods and values accumulated.
action reason	The reason an employee's job or employment information is updated. The action reason is entered in two parts: a personnel action, such as a promotion, termination, or change from one pay group to another—and a reason for that action. Action reasons are used by PeopleSoft Enterprise Human Resources, PeopleSoft Enterprise Benefits

	Administration, PeopleSoft Enterprise Stock Administration, and the COBRA Administration feature of the Base Benefits business process.
action template	In PeopleSoft Enterprise Receivables, outlines a set of escalating actions that the system or user performs based on the period of time that a customer or item has been in an action plan for a specific condition.
activity	<p>In PeopleSoft Enterprise Learning Management, an instance of a catalog item (sometimes called a class) that is available for enrollment. The activity defines such things as the costs that are associated with the offering, enrollment limits and deadlines, and waitlisting capacities.</p> <p>In PeopleSoft Enterprise Performance Management, the work of an organization and the aggregation of actions that are used for activity-based costing.</p> <p>In PeopleSoft Enterprise Project Costing, the unit of work that provides a further breakdown of projects—usually into specific tasks.</p> <p>In PeopleSoft Workflow, a specific transaction that you might need to perform in a business process. Because it consists of the steps that are used to perform a transaction, it is also known as a step map.</p>
address usage	In PeopleSoft Enterprise Campus Solutions, a grouping of address types defining the order in which the address types are used. For example, you might define an address usage code to process addresses in the following order: billing address, dormitory address, home address, and then work address.
adjustment calendar	In PeopleSoft Enterprise Campus Solutions, the adjustment calendar controls how a particular charge is adjusted on a student's account when the student drops classes or withdraws from a term. The charge adjustment is based on how much time has elapsed from a predetermined date, and it is determined as a percentage of the original charge amount.
administrative function	In PeopleSoft Enterprise Campus Solutions, a particular functional area that processes checklists, communication, and comments. The administrative function identifies which variable data is added to a person's checklist or communication record when a specific checklist code, communication category, or comment is assigned to the student. This key data enables you to trace that checklist, communication, or comment back to a specific processing event in a functional area.
admit type	In PeopleSoft Enterprise Campus Solutions, a designation used to distinguish first-year applications from transfer applications.
agreement	In PeopleSoft Enterprise eSettlements, provides a way to group and specify processing options, such as payment terms, pay from a bank, and notifications by a buyer and supplier location combination.
allocation rule	In PeopleSoft Enterprise Incentive Management, an expression within compensation plans that enables the system to assign transactions to nodes and participants. During transaction allocation, the allocation engine traverses the compensation structure from the current node to the root node, checking each node for plans that contain allocation rules.
alternate account	A feature in PeopleSoft Enterprise General Ledger that enables you to create a statutory chart of accounts and enter statutory account transactions at the detail transaction level, as required for recording and reporting by some national governments.
analysis database	In PeopleSoft Enterprise Campus Solutions, database tables that store large amounts of student information that may not appear in standard report formats. The analysis database tables contain keys for all objects in a report that an application program can use to reference other student-record objects that are not contained in the printed report. For instance, the analysis database contains data on courses that are considered

for satisfying a requirement but that are rejected. It also contains information on courses captured by global limits. An analysis database is used in PeopleSoft Enterprise Academic Advisement.

Application Messaging	PeopleSoft Application Messaging enables applications within the PeopleSoft Enterprise product family to communicate synchronously or asynchronously with other PeopleSoft Enterprise and third-party applications. An application message defines the records and fields to be published or subscribed to.
AR specialist	Abbreviation for <i>receivables specialist</i> . In PeopleSoft Enterprise Receivables, an individual in who tracks and resolves deductions and disputed items.
arbitration plan	In PeopleSoft Enterprise Pricer, defines how price rules are to be applied to the base price when the transaction is priced.
assessment rule	In PeopleSoft Enterprise Receivables, a user-defined rule that the system uses to evaluate the condition of a customer's account or of individual items to determine whether to generate a follow-up action.
asset class	An asset group used for reporting purposes. It can be used in conjunction with the asset category to refine asset classification.
attribute/value pair	In PeopleSoft Enterprise Directory Interface, relates the data that makes up an entry in the directory information tree.
audience	In PeopleSoft Enterprise Campus Solutions, a segment of the database that relates to an initiative, or a membership organization that is based on constituent attributes rather than a dues-paying structure. Examples of audiences include the Class of '65 and Undergraduate Arts & Sciences.
authentication server	A server that is set up to verify users of the system.
base time period	In PeopleSoft Enterprise Business Planning, the lowest level time period in a calendar.
benchmark job	In PeopleSoft Enterprise Workforce Analytics Solution, a benchmark job is a job code for which there is corresponding salary survey data from published, third-party sources.
billing career	In PeopleSoft Enterprise Campus Solutions, the one career under which other careers are grouped for billing purposes if a student is active simultaneously in multiple careers.
bio bit or bio brief	In PeopleSoft Enterprise Campus Solutions, a report that summarizes information stored in the system about a particular constituent. You can generate standard or specialized reports.
book	In PeopleSoft Enterprise Asset Management, used for storing financial and tax information, such as costs, depreciation attributes, and retirement information on assets.
branch	A tree node that rolls up to nodes above it in the hierarchy, as defined in PeopleSoft Tree Manager.
budgetary account only	An account used by the system only and not by users; this type of account does not accept transactions. You can only budget with this account. Formerly called "system-maintained account."
budget check	In commitment control, the processing of source transactions against control budget ledgers, to see if they pass, fail, or pass with a warning.
budget control	In commitment control, budget control ensures that commitments and expenditures don't exceed budgets. It enables you to track transactions against corresponding budgets and terminate a document's cycle if the defined budget conditions are not met.

For example, you can prevent a purchase order from being dispatched to a vendor if there are insufficient funds in the related budget to support it.

budget period	The interval of time (such as 12 months or 4 quarters) into which a period is divided for budgetary and reporting purposes. The ChartField allows maximum flexibility to define operational accounting time periods without restriction to only one calendar.
business activity	The name of a subset of a detailed business process. This might be a specific transaction, task, or action that you perform in a business process.
business event	In PeopleSoft Enterprise Receivables, defines the processing characteristics for the Receivable Update process for a draft activity. In PeopleSoft Enterprise Sales Incentive Management, an original business transaction or activity that may justify the creation of a PeopleSoft Enterprise Incentive Management event (a sale, for example).
business process	A standard set of 17 business processes are defined and maintained by the PeopleSoft Enterprise product families and are supported by the Business Process Engineering group. An example of a business process is Order Fulfillment, which is a business process that manages sales orders and contracts, inventory, billing, and so forth. See also <i>detailed business process</i> .
business task	The name of the specific function depicted in one of the business processes.
business unit	A corporation or a subset of a corporation that is independent with regard to one or more operational or accounting functions.
buyer	In PeopleSoft Enterprise eSettlements, an organization (or business unit, as opposed to an individual) that transacts with suppliers (vendors) within the system. A buyer creates payments for purchases that are made in the system.
campus	In PeopleSoft Enterprise Campus Solutions, an entity that is usually associated with a distinct physical administrative unit, that belongs to a single academic institution, that uses a unique course catalog, and that produces a common transcript for students within the same academic career.
catalog item	In PeopleSoft Enterprise Learning Management, a specific topic that a learner can study and have tracked. For example, "Introduction to Microsoft Word." A catalog item contains general information about the topic and includes a course code, description, categorization, keywords, and delivery methods. A catalog item can have one or more learning activities.
catalog map	In PeopleSoft Enterprise Catalog Management, translates values from the catalog source data to the format of the company's catalog.
catalog partner	In PeopleSoft Enterprise Catalog Management, shares responsibility with the enterprise catalog manager for maintaining catalog content.
categorization	Associates partner offerings with catalog offerings and groups them into enterprise catalog categories.
category	In PeopleSoft Enterprise Campus Solutions, a broad grouping to which specific comments or communications (contexts) are assigned. Category codes are also linked to 3C access groups so that you can assign data-entry or view-only privileges across functions.
channel	In PeopleSoft MultiChannel Framework, email, chat, voice (computer telephone integration [CTI]), or a generic event.
ChartField	A field that stores a chart of accounts, resources, and so on, depending on the PeopleSoft Enterprise application. ChartField values represent individual account numbers, department codes, and so forth.

ChartField balancing	You can require specific ChartFields to match up (balance) on the debit and the credit side of a transaction.
ChartField combination edit	The process of editing journal lines for valid ChartField combinations based on user-defined rules.
ChartKey	One or more fields that uniquely identify each row in a table. Some tables contain only one field as the key, while others require a combination.
checkbook	In PeopleSoft Enterprise Promotions Management, enables you to view financial data (such as planned, incurred, and actual amounts) that is related to funds and trade promotions.
checklist code	In PeopleSoft Enterprise Campus Solutions, a code that represents a list of planned or completed action items that can be assigned to a staff member, volunteer, or unit. Checklists enable you to view all action assignments on one page.
class	In PeopleSoft Enterprise Campus Solutions, a specific offering of a course component within an academic term. See also <i>course</i> .
Class ChartField	A ChartField value that identifies a unique appropriation budget key when you combine it with a fund, department ID, and program code, as well as a budget period. Formerly called <i>sub-classification</i> .
clearance	In PeopleSoft Enterprise Campus Solutions, the period of time during which a constituent in PeopleSoft Enterprise Contributor Relations is approved for involvement in an initiative or an action. Clearances are used to prevent development officers from making multiple requests to a constituent during the same time period.
clone	In PeopleCode, to make a unique copy. In contrast, to <i>copy</i> may mean making a new reference to an object, so if the underlying object is changed, both the copy and the original change.
cohort	In PeopleSoft Enterprise Campus Solutions, the highest level of the three-level classification structure that you define for enrollment management. You can define a cohort level, link it to other levels, and set enrollment target numbers for it. See also <i>population</i> and <i>division</i> .
collection	To make a set of documents available for searching in Verity, you must first create at least one collection. A collection is set of directories and files that allow search application users to use the Verity search engine to quickly find and display source documents that match search criteria. A collection is a set of statistics and pointers to the source documents, stored in a proprietary format on a file server. Because a collection can only store information for a single location, PeopleTools maintains a set of collections (one per language code) for each search index object.
collection rule	In PeopleSoft Enterprise Receivables, a user-defined rule that defines actions to take for a customer based on both the amount and the number of days past due for outstanding balances.
comm key	See <i>communication key</i> .
communication key	In PeopleSoft Enterprise Campus Solutions, a single code for entering a combination of communication category, communication context, communication method, communication direction, and standard letter code. Communication keys (also called <i>comm keys</i> or <i>speed keys</i>) can be created for background processes as well as for specific users.

compensation object	In PeopleSoft Enterprise Incentive Management, a node within a compensation structure. Compensation objects are the building blocks that make up a compensation structure's hierarchical representation.
compensation structure	In PeopleSoft Enterprise Incentive Management, a hierarchical relationship of compensation objects that represents the compensation-related relationship between the objects.
component interface	A component interface is a set of application programming interfaces (APIs) that you can use to access and modify PeopleSoft Enterprise database information using a program instead of the PeopleSoft client.
condition	In PeopleSoft Enterprise Receivables, occurs when there is a change of status for a customer's account, such as reaching a credit limit or exceeding a user-defined balance due.
configuration parameter catalog	Used to configure an external system with PeopleSoft Enterprise. For example, a configuration parameter catalog might set up configuration and communication parameters for an external server.
configuration plan	In PeopleSoft Enterprise Incentive Management, configuration plans hold allocation information for common variables (not incentive rules) and are attached to a node without a participant. Configuration plans are not processed by transactions.
constituents	In PeopleSoft Enterprise Campus Solutions, friends, alumni, organizations, foundations, or other entities affiliated with the institution, and about which the institution maintains information. The constituent types delivered with PeopleSoft Enterprise Contributor Relations Solutions are based on those defined by the Council for the Advancement and Support of Education (CASE).
content reference	Content references are pointers to content registered in the portal registry. These are typically either URLs or iScripts. Content references fall into three categories: target content, templates, and template pagelets.
context	In PeopleCode, determines which buffer fields can be contextually referenced and which is the current row of data on each scroll level when a PeopleCode program is running. In PeopleSoft Enterprise Campus Solutions, a specific instance of a comment or communication. One or more contexts are assigned to a category, which you link to 3C access groups so that you can assign data-entry or view-only privileges across functions. In PeopleSoft Enterprise Incentive Management, a mechanism that is used to determine the scope of a processing run. PeopleSoft Enterprise Incentive Management uses three types of context: plan, period, and run-level.
control table	Stores information that controls the processing of an application. This type of processing might be consistent throughout an organization, or it might be used only by portions of the organization for more limited sharing of data.
cost-plus contract line	A rate-based contract line associated with a fee component of Award, Fixed, Incentive, or Other. Rate-based contract lines associated with a fee type of None are not considered cost-plus contract lines.
cost profile	A combination of a receipt cost method, a cost flow, and a deplete cost method. A profile is associated with a cost book and determines how items in that book are valued, as well as how the material movement of the item is valued for the book.
cost row	A cost transaction and amount for a set of ChartFields.
course	In PeopleSoft Enterprise Campus Solutions, a course that is offered by a school and that is typically described in a course catalog. A course has a standard syllabus and

credit level; however, these may be modified at the class level. Courses can contain multiple components such as lecture, discussion, and lab.

See also *class*.

course share set	In PeopleSoft Enterprise Campus Solutions, a tag that defines a set of requirement groups that can share courses. Course share sets are used in PeopleSoft Enterprise Academic Advisement.
current learning	In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's in-progress learning activities and programs.
data acquisition	In PeopleSoft Enterprise Incentive Management, the process during which raw business transactions are acquired from external source systems and fed into the operational data store (ODS).
data cube	In PeopleSoft Analytic Calculation Engine, a data cube is a container for one kind of data (such as Sales data) and works with in tandem with one or more dimensions. Dimensions and data cubes in PeopleSoft Analytic Calculation Engine are unrelated to dimensions and online analytical processing (OLAP) cubes in PeopleSoft Cube Manager.
data elements	Data elements, at their simplest level, define a subset of data and the rules by which to group them. For Workforce Analytics, data elements are rules that tell the system what measures to retrieve about your workforce groups.
dataset	A data grouping that enables role-based filtering and distribution of data. You can limit the range and quantity of data that is displayed for a user by associating dataset rules with user roles. The result of dataset rules is a set of data that is appropriate for the user's roles.
delivery method	In PeopleSoft Enterprise Learning Management, identifies the primary type of delivery method in which a particular learning activity is offered. Also provides default values for the learning activity, such as cost and language. This is primarily used to help learners search the catalog for the type of delivery from which they learn best. Because PeopleSoft Enterprise Learning Management is a blended learning system, it does not enforce the delivery method. In PeopleSoft Enterprise Supply Chain Management, identifies the method by which goods are shipped to their destinations (such as truck, air, and rail). The delivery method is specified when creating shipment schedules.
delivery method type	In PeopleSoft Enterprise Learning Management, identifies how learning activities can be delivered—for example, through online learning, classroom instruction, seminars, books, and so forth—in an organization. The type determines whether the delivery method includes scheduled components.
detailed business process	A subset of the business process. For example, the detailed business process named Determine Cash Position is a subset of the business process called Cash Management.
dimension	In PeopleSoft Analytic Calculation Engine, a dimension contains a list of one kind of data that can span various contexts, and it is a basic component of an analytic model. Within the analytic model, a dimension is attached to one or more data cubes. In PeopleSoft Cube Manager, a dimension is the most basic component of an OLAP cube and specifies the PeopleSoft metadata to be used to create the dimension's rollup structure. Dimensions and data cubes in PeopleSoft Analytic Calculation Engine are unrelated to dimensions and OLAP cubes in PeopleSoft Cube Manager.
directory information tree	In PeopleSoft Enterprise Directory Interface, the representation of a directory's hierarchical structure.

division	In PeopleSoft Enterprise Campus Solutions, the lowest level of the three-level classification structure that you define in PeopleSoft Enterprise Recruiting and Admissions for enrollment management. You can define a division level, link it to other levels, and set enrollment target numbers for it. See also <i>population</i> and <i>cohort</i> .
document sequencing	A flexible method that sequentially numbers the financial transactions (for example, bills, purchase orders, invoices, and payments) in the system for statutory reporting and for tracking commercial transaction activity.
dynamic detail tree	A tree that takes its detail values—dynamic details—directly from a table in the database, rather than from a range of values that are entered by the user.
edit table	A table in the database that has its own record definition, such as the Department table. As fields are entered into a PeopleSoft Enterprise application, they can be validated against an edit table to ensure data integrity throughout the system.
effective date	A method of dating information in PeopleSoft Enterprise applications. You can predate information to add historical data to your system, or postdate information in order to enter it before it actually goes into effect. By using effective dates, you don't delete values; you enter a new value with a current effective date.
EIM ledger	Abbreviation for <i>Enterprise Incentive Management ledger</i> . In PeopleSoft Enterprise Incentive Management, an object to handle incremental result gathering within the scope of a participant. The ledger captures a result set with all of the appropriate traces to the data origin and to the processing steps of which it is a result.
elimination set	In PeopleSoft Enterprise General Ledger, a related group of intercompany accounts that is processed during consolidations.
entry event	In PeopleSoft Enterprise General Ledger, Receivables, Payables, Purchasing, and Billing, a business process that generates multiple debits and credits resulting from single transactions to produce standard, supplemental accounting entries.
equitization	In PeopleSoft Enterprise General Ledger, a business process that enables parent companies to calculate the net income of subsidiaries on a monthly basis and adjust that amount to increase the investment amount and equity income amount before performing consolidations.
equity item limit	In PeopleSoft Enterprise Campus Solutions, the amounts of funds set by the institution to be awarded with discretionary or gift funds. The limit could be reduced by amounts equal to such things as expected family contribution (EFC) or parent contribution. Students are packaged by Equity Item Type Groups and Related Equity Item Types. This limit can be used to assure that similar student populations are packaged equally.
event	A predefined point either in the Component Processor flow or in the program flow. As each point is encountered, the event activates each component, triggering any PeopleCode program that is associated with that component and that event. Examples of events are FieldChange, SavePreChange, and RowDelete. In PeopleSoft Enterprise Human Resources, also refers to an incident that affects benefits eligibility.
event propagation process	In PeopleSoft Enterprise Sales Incentive Management, a process that determines, through logic, the propagation of an original PeopleSoft Enterprise Incentive Management event and creates a derivative (duplicate) of the original event to be processed by other objects. PeopleSoft Enterprise Enterprise Sales Incentive Management uses this mechanism to implement splits, roll-ups, and so on. Event propagation determines who receives the credit.
exception	In PeopleSoft Enterprise Receivables, an item that either is a deduction or is in dispute.

exclusive pricing	In PeopleSoft Enterprise Order Management, a type of arbitration plan that is associated with a price rule. Exclusive pricing is used to price sales order transactions.
fact	In PeopleSoft Enterprise applications, facts are numeric data values from fields from a source database as well as an analytic application. A fact can be anything you want to measure your business by, for example, revenue, actual, budget data, or sales numbers. A fact is stored on a fact table.
financial aid term	In PeopleSoft Enterprise Campus Solutions, a combination of a period of time that the school determines as an instructional accounting period and an academic career. It is created and defined during the setup process. Only terms eligible for financial aid are set up for each financial aid career.
forecast item	A logical entity with a unique set of descriptive demand and forecast data that is used as the basis to forecast demand. You create forecast items for a wide range of uses, but they ultimately represent things that you buy, sell, or use in your organization and for which you require a predictable usage.
fund	In PeopleSoft Enterprise Promotions Management, a budget that can be used to fund promotional activity. There are four funding methods: top down, fixed accrual, rolling accrual, and zero-based accrual.
gap	In PeopleSoft Enterprise Campus Solutions, an artificial figure that sets aside an amount of unmet financial aid need that is not funded with Title IV funds. A gap can be used to prevent fully funding any student to conserve funds, or it can be used to preserve unmet financial aid need so that institutional funds can be awarded.
generic process type	In PeopleSoft Process Scheduler, process types are identified by a generic process type. For example, the generic process type SQR includes all SQR process types, such as SQR process and SQR report.
gift table	In PeopleSoft Enterprise Campus Solutions, a table or so-called <i>donor pyramid</i> describing the number and size of gifts that you expect will be needed to successfully complete the campaign in PeopleSoft Enterprise Contributor Relations. The gift table enables you to estimate the number of donors and prospects that you need at each gift level to reach the campaign goal.
GL business unit	Abbreviation for <i>general ledger business unit</i> . A unit in an organization that is an independent entity for accounting purposes. It maintains its own set of accounting books. See also <i>business unit</i> .
GL entry template	Abbreviation for <i>general ledger entry template</i> . In PeopleSoft Enterprise Campus Solutions, a template that defines how a particular item is sent to the general ledger. An item-type maps to the general ledger, and the GL entry template can involve multiple general ledger accounts. The entry to the general ledger is further controlled by high-level flags that control the summarization and the type of accounting—that is, accrual or cash.
GL Interface process	Abbreviation for <i>General Ledger Interface process</i> . In PeopleSoft Enterprise Campus Solutions, a process that is used to send transactions from PeopleSoft Enterprise Student Financials to the general ledger. Item types are mapped to specific general ledger accounts, enabling transactions to move to the general ledger when the GL Interface process is run.
group	In PeopleSoft Enterprise Billing and Receivables, a posting entity that comprises one or more transactions (items, deposits, payments, transfers, matches, or write-offs). In PeopleSoft Enterprise Human Resources Management and Supply Chain Management, any set of records that are associated under a single name or variable to

	run calculations in PeopleSoft business processes. In PeopleSoft Enterprise Time and Labor, for example, employees are placed in groups for time reporting purposes.
incentive object	In PeopleSoft Enterprise Incentive Management, the incentive-related objects that define and support the PeopleSoft Enterprise Incentive Management calculation process and results, such as plan templates, plans, results data, and user interaction objects.
incentive rule	In PeopleSoft Enterprise Sales Incentive Management, the commands that act on transactions and turn them into compensation. A rule is one part in the process of turning a transaction into compensation.
incur	In PeopleSoft Enterprise Promotions Management, to become liable for a promotional payment. In other words, you owe that amount to a customer for promotional activities.
initiative	In PeopleSoft Enterprise Campus Solutions, the basis from which all advancement plans are executed. It is an organized effort targeting a specific constituency, and it can occur over a specified period of time with specific purposes and goals. An initiative can be a campaign, an event, an organized volunteer effort, a membership drive, or any other type of effort defined by the institution. Initiatives can be multipart, and they can be related to other initiatives. This enables you to track individual parts of an initiative, as well as entire initiatives.
inquiry access	In PeopleSoft Enterprise Campus Solutions, a type of security access that permits the user only to view data. See also <i>update access</i> .
institution	In PeopleSoft Enterprise Campus Solutions, an entity (such as a university or college) that is independent of other similar entities and that has its own set of rules and business processes.
integration	A relationship between two compatible integration points that enables communication to take place between systems. Integrations enable PeopleSoft Enterprise applications to work seamlessly with other PeopleSoft Enterprise applications or with third-party systems or software.
integration point	An interface that a system uses to communicate with another PeopleSoft Enterprise application or an external application.
integration set	A logical grouping of integrations that applications use for the same business purpose. For example, the integration set <code>ADVANCED_SHIPPING_ORDER</code> contains all of the integrations that notify a customer that an order has shipped.
item	In PeopleSoft Enterprise Inventory, a tangible commodity that is stored in a business unit (shipped from a warehouse). In PeopleSoft Enterprise Demand Planning, Inventory Policy Planning, and Supply Planning, a noninventory item that is designated as being used for planning purposes only. It can represent a family or group of inventory items. It can have a planning bill of material (BOM) or planning routing, and it can exist as a component on a planning BOM. A planning item cannot be specified on a production or engineering BOM or routing, and it cannot be used as a component in a production. The quantity on hand will never be maintained.
	In PeopleSoft Enterprise Receivables, an individual receivable. An item can be an invoice, a credit memo, a debit memo, a write-off, or an adjustment.
item shuffle	In PeopleSoft Enterprise Campus Solutions, a process that enables you to change a payment allocation without having to reverse the payment.

joint communication	In PeopleSoft Enterprise Campus Solutions, one letter that is addressed jointly to two people. For example, a letter might be addressed to both Mr. Sudhir Awat and Ms. Samantha Mortelli. A relationship must be established between the two individuals in the database, and at least one of the individuals must have an ID in the database.
keyword	In PeopleSoft Enterprise Campus Solutions, a term that you link to particular elements within PeopleSoft Enterprise Student Financials, Financial Aid, and Contributor Relations. You can use keywords as search criteria that enable you to locate specific records in a search dialog box.
KPI	An abbreviation for <i>key performance indicator</i> . A high-level measurement of how well an organization is doing in achieving critical success factors. This defines the data value or calculation upon which an assessment is determined.
LDIF file	Abbreviation for <i>Lightweight Directory Access Protocol (LDAP) Data Interchange Format file</i> . Contains discrepancies between PeopleSoft Enterprise data and directory data.
learner group	In PeopleSoft Enterprise Learning Management, a group of learners who are linked to the same learning environment. Members of the learner group can share the same attributes, such as the same department or job code. Learner groups are used to control access to and enrollment in learning activities and programs. They are also used to perform group enrollments and mass enrollments in the back office.
learning components	In PeopleSoft Enterprise Learning Management, the foundational building blocks of learning activities. PeopleSoft Enterprise Learning Management supports six basic types of learning components: web-based, session, webcast, test, survey, and assignment. One or more of these learning component types compose a single learning activity.
learning environment	In PeopleSoft Enterprise Learning Management, identifies a set of categories and catalog items that can be made available to learner groups. Also defines the default values that are assigned to the learning activities and programs that are created within a particular learning environment. Learning environments provide a way to partition the catalog so that learners see only those items that are relevant to them.
learning history	In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's completed learning activities and programs.
ledger mapping	You use ledger mapping to relate expense data from general ledger accounts to resource objects. Multiple ledger line items can be mapped to one or more resource IDs. You can also use ledger mapping to map dollar amounts (referred to as <i>rates</i>) to business units. You can map the amounts in two different ways: an actual amount that represents actual costs of the accounting period, or a budgeted amount that can be used to calculate the capacity rates as well as budgeted model results. In PeopleSoft Enterprise Warehouse, you can map general ledger accounts to the EW Ledger table.
library section	In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan (or template) and that is available for other plans to share. Changes to a library section are reflected in all plans that use it.
linked section	In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan template but appears in a plan. Changes to linked sections propagate to plans using that section.
linked variable	In PeopleSoft Enterprise Incentive Management, a variable that is defined and maintained in a plan template and that also appears in a plan. Changes to linked variables propagate to plans using that variable.
LMS	Abbreviation for <i>learning management system</i> . In PeopleSoft Enterprise Campus Solutions, LMS is a PeopleSoft Enterprise Student Records feature that provides a

common set of interoperability standards that enable the sharing of instructional content and data between learning and administrative environments.

load	In PeopleSoft Enterprise Inventory, identifies a group of goods that are shipped together. Load management is a feature of PeopleSoft Enterprise Inventory that is used to track the weight, the volume, and the destination of a shipment.
local functionality	In PeopleSoft Enterprise HRMS, the set of information that is available for a specific country. You can access this information when you click the appropriate country flag in the global window, or when you access it by a local country menu.
location	Locations enable you to indicate the different types of addresses—for a company, for example, one address to receive bills, another for shipping, a third for postal deliveries, and a separate street address. Each address has a different location number. The primary location—indicated by a <i>1</i> —is the address you use most often and may be different from the main address.
logistical task	In PeopleSoft Enterprise Services Procurement, an administrative task that is related to hiring a service provider. Logistical tasks are linked to the service type on the work order so that different types of services can have different logistical tasks. Logistical tasks include both preapproval tasks (such as assigning a new badge or ordering a new laptop) and postapproval tasks (such as scheduling orientation or setting up the service provider email). The logistical tasks can be mandatory or optional. Mandatory preapproval tasks must be completed before the work order is approved. Mandatory postapproval tasks, on the other hand, must be completed before a work order is released to a service provider.
market template	In PeopleSoft Enterprise Incentive Management, additional functionality that is specific to a given market or industry and is built on top of a product category.
mass change	In PeopleSoft Enterprise Campus Solutions, mass change is a SQL generator that can be used to create specialized functionality. Using mass change, you can set up a series of Insert, Update, or Delete SQL statements to perform business functions that are specific to the institution. See also <i>3C engine</i> .
match group	In PeopleSoft Enterprise Receivables, a group of receivables items and matching offset items. The system creates match groups by using user-defined matching criteria for selected field values.
MCF server	Abbreviation for <i>PeopleSoft MultiChannel Framework server</i> . Comprises the universal queue server and the MCF log server. Both processes are started when <i>MCF Servers</i> is selected in an application server domain configuration.
merchandising activity	In PeopleSoft Enterprise Promotions Management, a specific discount type that is associated with a trade promotion (such as off-invoice, billback or rebate, or lump-sum payment) that defines the performance that is required to receive the discount. In the industry, you may know this as an offer, a discount, a merchandising event, an event, or a tactic.
meta-SQL	Meta-SQL constructs expand into platform-specific SQL substrings. They are used in functions that pass SQL strings, such as in SQL objects, the <i>SQLExec</i> function, and PeopleSoft Application Engine programs.
metastring	Metastrings are special expressions included in SQL string literals. The metastrings, prefixed with a percent (%) symbol, are included directly in the string literals. They expand at run time into an appropriate substring for the current database platform.
multibook	In PeopleSoft Enterprise General Ledger, multiple ledgers having multiple-base currencies that are defined for a business unit, with the option to post a single

	transaction to all base currencies (all ledgers) or to only one of those base currencies (ledgers).
multicurrency	The ability to process transactions in a currency other than the business unit's base currency.
national allowance	In PeopleSoft Enterprise Promotions Management, a promotion at the corporate level that is funded by nondiscretionary dollars. In the industry, you may know this as a national promotion, a corporate promotion, or a corporate discount.
need	In PeopleSoft Enterprise Campus Solutions, the difference between the cost of attendance (COA) and the expected family contribution (EFC). It is the gap between the cost of attending the school and the student's resources. The financial aid package is based on the amount of financial need. The process of determining a student's need is called <i>need analysis</i> .
node-oriented tree	A tree that is based on a detail structure, but the detail values are not used.
pagelet	Each block of content on the home page is called a pagelet. These pagelets display summary information within a small rectangular area on the page. The pagelet provide users with a snapshot of their most relevant PeopleSoft Enterprise and non-PeopleSoft Enterprise content.
participant	In PeopleSoft Enterprise Incentive Management, participants are recipients of the incentive compensation calculation process.
participant object	Each participant object may be related to one or more compensation objects. See also <i>compensation object</i> .
partner	A company that supplies products or services that are resold or purchased by the enterprise.
pay cycle	In PeopleSoft Enterprise Payables, a set of rules that define the criteria by which it should select scheduled payments for payment creation.
payment shuffle	In PeopleSoft Enterprise Campus Solutions, a process allowing payments that have been previously posted to a student's account to be automatically reapplied when a higher priority payment is posted or the payment allocation definition is changed.
pending item	In PeopleSoft Enterprise Receivables, an individual receivable (such as an invoice, a credit memo, or a write-off) that has been entered in or created by the system, but hasn't been posted.
PeopleCode	PeopleCode is a proprietary language, executed by the PeopleSoft Enterprise component processor. PeopleCode generates results based on existing data or user actions. By using various tools provided with PeopleTools, external services are available to all PeopleSoft Enterprise applications wherever PeopleCode can be executed.
PeopleCode event	See <i>event</i> .
PeopleSoft Pure Internet Architecture	The fundamental architecture on which PeopleSoft 8 applications are constructed, consisting of a relational database management system (RDBMS), an application server, a web server, and a browser.
performance measurement	In PeopleSoft Enterprise Incentive Management, a variable used to store data (similar to an aggregator, but without a predefined formula) within the scope of an incentive plan. Performance measures are associated with a plan calendar, territory, and participant. Performance measurements are used for quota calculation and reporting.
period context	In PeopleSoft Enterprise Incentive Management, because a participant typically uses the same compensation plan for multiple periods, the period context associates

a plan context with a specific calendar period and fiscal year. The period context references the associated plan context, thus forming a chain. Each plan context has a corresponding set of period contexts.

person of interest	A person about whom the organization maintains information but who is not part of the workforce.
personal portfolio	In PeopleSoft Enterprise Campus Solutions, the user-accessible menu item that contains an individual's name, address, telephone number, and other personal information.
plan	In PeopleSoft Enterprise Sales Incentive Management, a collection of allocation rules, variables, steps, sections, and incentive rules that instruct the PeopleSoft Enterprise Incentive Management engine in how to process transactions.
plan context	In PeopleSoft Enterprise Incentive Management, correlates a participant with the compensation plan and node to which the participant is assigned, enabling the PeopleSoft Enterprise Incentive Management system to find anything that is associated with the node and that is required to perform compensation processing. Each participant, node, and plan combination represents a unique plan context—if three participants are on a compensation structure, each has a different plan context. Configuration plans are identified by plan contexts and are associated with the participants that refer to them.
plan template	In PeopleSoft Enterprise Incentive Management, the base from which a plan is created. A plan template contains common sections and variables that are inherited by all plans that are created from the template. A template may contain steps and sections that are not visible in the plan definition.
planned learning	In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's planned learning activities and programs.
planning instance	In PeopleSoft Enterprise Supply Planning, a set of data (business units, items, supplies, and demands) constituting the inputs and outputs of a supply plan.
population	In PeopleSoft Enterprise Campus Solutions, the middle level of the three-level classification structure that you define in PeopleSoft Enterprise Recruiting and Admissions for enrollment management. You can define a population level, link it to other levels, and set enrollment target numbers for it. See also <i>division</i> and <i>cohort</i> .
portal registry	In PeopleSoft Enterprise applications, the portal registry is a tree-like structure in which content references are organized, classified, and registered. It is a central repository that defines both the structure and content of a portal through a hierarchical, tree-like structure of folders useful for organizing and securing content references.
price list	In PeopleSoft Enterprise Pricer, enables you to select products and conditions for which the price list applies to a transaction. During a transaction, the system either determines the product price based on the predefined search hierarchy for the transaction or uses the product's lowest price on any associated, active price lists. This price is used as the basis for any further discounts and surcharges.
price rule	In PeopleSoft Enterprise Pricer, defines the conditions that must be met for adjustments to be applied to the base price. Multiple rules can apply when conditions of each rule are met.
price rule condition	In PeopleSoft Enterprise Pricer, selects the price-by fields, the values for the price-by fields, and the operator that determines how the price-by fields are related to the transaction.
price rule key	In PeopleSoft Enterprise Pricer, defines the fields that are available to define price rule conditions (which are used to match a transaction) on the price rule.

primacy number	In PeopleSoft Enterprise Campus Solutions, a number that the system uses to prioritize financial aid applications when students are enrolled in multiple academic careers and academic programs at the same time. The Consolidate Academic Statistics process uses the primacy number indicated for both the career and program at the institutional level to determine a student's primary career and program. The system also uses the number to determine the primary student attribute value that is used when you extract data to report on cohorts. The lowest number takes precedence.
primary name type	In PeopleSoft Enterprise Campus Solutions, the name type that is used to link the name stored at the highest level within the system to the lower-level set of names that an individual provides.
process category	In PeopleSoft Process Scheduler, processes that are grouped for server load balancing and prioritization.
process group	In PeopleSoft Enterprise Financials, a group of application processes (performed in a defined order) that users can initiate in real time, directly from a transaction entry page.
process definition	Process definitions define each run request.
process instance	A unique number that identifies each process request. This value is automatically incremented and assigned to each requested process when the process is submitted to run.
process job	You can link process definitions into a job request and process each request serially or in parallel. You can also initiate subsequent processes based on the return code from each prior request.
process request	A single run request, such as a Structured Query Report (SQR), a COBOL or Application Engine program, or a Crystal report that you run through PeopleSoft Process Scheduler.
process run control	A PeopleTools variable used to retain PeopleSoft Process Scheduler values needed at runtime for all requests that reference a run control ID. Do not confuse these with application run controls, which may be defined with the same run control ID, but only contain information specific to a given application process request.
product	A PeopleSoft Enterprise or third-party product. PeopleSoft organizes its software products into product families and product lines. Interactive Services Repository contains information about every release of every product that PeopleSoft sells, as well as products from certified third-party companies. These products appear with the product name and release number.
product category	In PeopleSoft Enterprise Incentive Management, indicates an application in the PeopleSoft Enterprise Incentive Management suite of products. Each transaction in the PeopleSoft Enterprise Incentive Management system is associated with a product category.
product family	A group of products that are related by common functionality. The family names that can be searched using Interactive Service Repository are Oracle's PeopleSoft Enterprise, PeopleSoft EnterpriseOne, PeopleSoft World, and third-party, certified partners.
product line	The name of a PeopleSoft Enterprise product line or the company name of a third-party certified partner. Integration Services Repository enables you to search for integration points by product line.
programs	In PeopleSoft Enterprise Learning Management, a high-level grouping that guides the learner along a specific learning path through sections of catalog items. PeopleSoft Enterprise Learning Systems provides two types of programs—curricula and certifications.

progress log	In PeopleSoft Enterprise Services Procurement, tracks deliverable-based projects. This is similar to the time sheet in function and process. The service provider contact uses the progress log to record and submit progress on deliverables. The progress can be logged by the activity that is performed, by the percentage of work that is completed, or by the completion of milestone activities that are defined for the project.
project transaction	In PeopleSoft Enterprise Project Costing, an individual transaction line that represents a cost, time, budget, or other transaction row.
promotion	In PeopleSoft Enterprise Promotions Management, a trade promotion, which is typically funded from trade dollars and used by consumer products manufacturers to increase sales volume.
prospects	In PeopleSoft Enterprise Campus Solutions, students who are interested in applying to the institution. In PeopleSoft Enterprise Contributor Relations, individuals and organizations that are most likely to make substantial financial commitments or other types of commitments to the institution.
publishing	In PeopleSoft Enterprise Incentive Management, a stage in processing that makes incentive-related results available to participants.
rating components	In PeopleSoft Enterprise Campus Solutions, variables used with the Equation Editor to retrieve specified populations.
record group	A set of logically and functionally related control tables and views. Record groups help enable TableSet sharing, which eliminates redundant data entry. Record groups ensure that TableSet sharing is applied consistently across all related tables and views.
record input VAT flag	Abbreviation for <i>record input value-added tax flag</i> . Within PeopleSoft Enterprise Purchasing, Payables, and General Ledger, this flag indicates that you are recording input VAT on the transaction. This flag, in conjunction with the record output VAT flag, is used to determine the accounting entries created for a transaction and to determine how a transaction is reported on the VAT return. For all cases within Purchasing and Payables where VAT information is tracked on a transaction, this flag is set to Yes. This flag is not used in PeopleSoft Enterprise Order Management, Billing, or Receivables, where it is assumed that you are always recording only output VAT, or in PeopleSoft Enterprise Expenses, where it is assumed that you are always recording only input VAT.
record output VAT flag	Abbreviation for <i>record output value-added tax flag</i> . See <i>record input VAT flag</i> .
recname	The name of a record that is used to determine the associated field to match a value or set of values.
recognition	In PeopleSoft Enterprise Campus Solutions, the recognition type indicates whether the PeopleSoft Enterprise Contributor Relations donor is the primary donor of a commitment or shares the credit for a donation. Primary donors receive hard credit that must total 100 percent. Donors that share the credit are given soft credit. Institutions can also define other share recognition-type values such as memo credit or vehicle credit.
reference data	In PeopleSoft Enterprise Sales Incentive Management, system objects that represent the sales organization, such as territories, participants, products, customers, and channels.
reference object	In PeopleSoft Enterprise Incentive Management, this dimension-type object further defines the business. Reference objects can have their own hierarchy (for example, product tree, customer tree, industry tree, and geography tree).

reference transaction	In commitment control, a reference transaction is a source transaction that is referenced by a higher-level (and usually later) source transaction, in order to automatically reverse all or part of the referenced transaction's budget-checked amount. This avoids duplicate postings during the sequential entry of the transaction at different commitment levels. For example, the amount of an encumbrance transaction (such as a purchase order) will, when checked and recorded against a budget, cause the system to concurrently reference and relieve all or part of the amount of a corresponding pre-encumbrance transaction, such as a purchase requisition.
regional sourcing	In PeopleSoft Enterprise Purchasing, provides the infrastructure to maintain, display, and select an appropriate vendor and vendor pricing structure that is based on a regional sourcing model where the multiple ship to locations are grouped. Sourcing may occur at a level higher than the ship to location.
relationship object	In PeopleSoft Enterprise Incentive Management, these objects further define a compensation structure to resolve transactions by establishing associations between compensation objects and business objects.
remote data source data	Data that is extracted from a separate database and migrated into the local database.
REN server	Abbreviation for <i>real-time event notification server</i> in PeopleSoft MultiChannel Framework.
requester	In PeopleSoft Enterprise eSettlements, an individual who requests goods or services and whose ID appears on the various procurement pages that reference purchase orders.
reversal indicator	In PeopleSoft Enterprise Campus Solutions, an indicator that denotes when a particular payment has been reversed, usually because of insufficient funds.
role	Describes how people fit into PeopleSoft Workflow. A role is a class of users who perform the same type of work, such as clerks or managers. Your business rules typically specify what user role needs to do an activity.
role user	A PeopleSoft Workflow user. A person's role user ID serves much the same purpose as a user ID does in other parts of the system. PeopleSoft Workflow uses role user IDs to determine how to route worklist items to users (through an email address, for example) and to track the roles that users play in the workflow. Role users do not need PeopleSoft user IDs.
roll up	In a tree, to roll up is to total sums based on the information hierarchy.
run control	A run control is a type of online page that is used to begin a process, such as the batch processing of a payroll run. Run control pages generally start a program that manipulates data.
run control ID	A unique ID to associate each user with his or her own run control table entries.
run-level context	In PeopleSoft Enterprise Incentive Management, associates a particular run (and batch ID) with a period context and plan context. Every plan context that participates in a run has a separate run-level context. Because a run cannot span periods, only one run-level context is associated with each plan context.
SCP SCBM XML message	Abbreviation for <i>Supply Chain Planning Supply Chain Business Modeler Extensible Markup Language message</i> . Supply Chain Business Modeler uses XML as the format for all data that it imports and exports.
search query	You use this set of objects to pass a query string and operators to the search engine. The search index returns a set of matching results with keys to the source documents.
search/match	In PeopleSoft Enterprise Campus Solutions and PeopleSoft Enterprise Human Resources Management Solutions, a feature that enables you to search for and identify duplicate records in the database.

seasonal address	In PeopleSoft Enterprise Campus Solutions, an address that recurs for the same length of time at the same time of year each year until adjusted or deleted.
section	In PeopleSoft Enterprise Incentive Management, a collection of incentive rules that operate on transactions of a specific type. Sections enable plans to be segmented to process logical events in different sections.
security event	In commitment control, security events trigger security authorization checking, such as budget entries, transfers, and adjustments; exception overrides and notifications; and inquiries.
serial genealogy	In PeopleSoft Enterprise Manufacturing, the ability to track the composition of a specific, serial-controlled item.
serial in production	In PeopleSoft Enterprise Manufacturing, enables the tracing of serial information for manufactured items. This is maintained in the Item Master record.
service impact	In PeopleSoft Enterprise Campus Solutions, the resulting action triggered by a service indicator. For example, a service indicator that reflects nonpayment of account balances by a student might result in a service impact that prohibits registration for classes.
service indicator	In PeopleSoft Enterprise Campus Solutions, indicates services that may be either withheld or provided to an individual. Negative service indicators indicate holds that prevent the individual from receiving specified services, such as check-cashing privileges or registration for classes. Positive service indicators designate special services that are provided to the individual, such as front-of-line service or special services for disabled students.
session	<p>In PeopleSoft Enterprise Campus Solutions, time elements that subdivide a term into multiple time periods during which classes are offered. In PeopleSoft Enterprise Contributor Relations, a session is the means of validating gift, pledge, membership, or adjustment data entry . It controls access to the data entered by a specific user ID. Sessions are balanced, queued, and then posted to the institution's financial system. Sessions must be posted to enter a matching gift or pledge payment, to make an adjustment, or to process giving clubs or acknowledgements.</p> <p>In PeopleSoft Enterprise Learning Management, a single meeting day of an activity (that is, the period of time between start and finish times within a day). The session stores the specific date, location, meeting time, and instructor. Sessions are used for scheduled training.</p>
session template	In PeopleSoft Enterprise Learning Management, enables you to set up common activity characteristics that may be reused while scheduling a PeopleSoft Enterprise Learning Management activity—characteristics such as days of the week, start and end times, facility and room assignments, instructors, and equipment. A session pattern template can be attached to an activity that is being scheduled. Attaching a template to an activity causes all of the default template information to populate the activity session pattern.
setup relationship	In PeopleSoft Enterprise Incentive Management, a relationship object type that associates a configuration plan with any structure node.
share driver expression	In PeopleSoft Enterprise Business Planning, a named planning method similar to a driver expression, but which you can set up globally for shared use within a single planning application or to be shared between multiple planning applications through PeopleSoft Enterprise Warehouse.
single signon	With single signon, users can, after being authenticated by a PeopleSoft Enterprise application server, access a second PeopleSoft Enterprise application server without entering a user ID or password.

source key process	In PeopleSoft Enterprise Campus Solutions, a process that relates a particular transaction to the source of the charge or financial aid. On selected pages, you can drill down into particular charges.
source transaction	In commitment control, any transaction generated in a PeopleSoft Enterprise or third-party application that is integrated with commitment control and which can be checked against commitment control budgets. For example, a pre-encumbrance, encumbrance, expenditure, recognized revenue, or collected revenue transaction.
speed key	See <i>communication key</i> .
SpeedChart	A user-defined shorthand key that designates several ChartKeys to be used for voucher entry. Percentages can optionally be related to each ChartKey in a SpeedChart definition.
SpeedType	A code representing a combination of ChartField values. SpeedTypes simplify the entry of ChartFields commonly used together.
staging	A method of consolidating selected partner offerings with the offerings from the enterprise's other partners.
standard letter code	In PeopleSoft Enterprise Campus Solutions, a standard letter code used to identify each letter template available for use in mail merge functions. Every letter generated in the system must have a standard letter code identification.
statutory account	Account required by a regulatory authority for recording and reporting financial results. In PeopleSoft Enterprise, this is equivalent to the Alternate Account (ALTACCT) ChartField.
step	In PeopleSoft Enterprise Sales Incentive Management, a collection of sections in a plan. Each step corresponds to a step in the job run.
storage level	In PeopleSoft Enterprise Inventory, identifies the level of a material storage location. Material storage locations are made up of a business unit, a storage area, and a storage level. You can set up to four storage levels.
subcustomer qualifier	A value that groups customers into a division for which you can generate detailed history, aging, events, and profiles.
Summary ChartField	You use summary ChartFields to create summary ledgers that roll up detail amounts based on specific detail values or on selected tree nodes. When detail values are summarized using tree nodes, summary ChartFields must be used in the summary ledger data record to accommodate the maximum length of a node name (20 characters).
summary ledger	An accounting feature used primarily in allocations, inquiries, and PS/nVision reporting to store combined account balances from detail ledgers. Summary ledgers increase speed and efficiency of reporting by eliminating the need to summarize detail ledger balances each time a report is requested. Instead, detail balances are summarized in a background process according to user-specified criteria and stored on summary ledgers. The summary ledgers are then accessed directly for reporting.
summary time period	In PeopleSoft Enterprise Business Planning, any time period (other than a base time period) that is an aggregate of other time periods, including other summary time periods and base time periods, such as quarter and year total.
summary tree	A tree used to roll up accounts for each type of report in summary ledgers. Summary trees enable you to define trees on trees. In a summary tree, the detail values are really nodes on a detail tree or another summary tree (known as the <i>basis</i> tree). A summary tree structure specifies the details on which the summary trees are to be built.
syndicate	To distribute a production version of the enterprise catalog to partners.

system function	In PeopleSoft Enterprise Receivables, an activity that defines how the system generates accounting entries for the general ledger.
system source	<p>The system source identifies the source of a transaction row in the database. For example, a transaction that originates in PeopleSoft Enterprise Expenses contains a system source code of BEX (Expenses Batch).</p> <p>When PeopleSoft Enterprise Project Costing prices the source transaction row for billing, the system creates a new row with a system source code of PRP (Project Costing pricing), which represents the system source of the new row. System source codes can identify sources that are internal or external to the PeopleSoft Enterprise system. For example, processes that import data from Microsoft Project into PeopleSoft Enterprise applications create transaction rows with a source code of MSP (Microsoft Project).</p>
TableSet	A means of sharing similar sets of values in control tables, where the actual data values are different but the structure of the tables is the same.
TableSet sharing	Shared data that is stored in many tables that are based on the same TableSets. Tables that use TableSet sharing contain the SETID field as an additional key or unique identifier.
target currency	The value of the entry currency or currencies converted to a single currency for budget viewing and inquiry purposes.
tax authority	In PeopleSoft Enterprise Campus Solutions, a user-defined element that combines a description and percentage of a tax with an account type, an item type, and a service impact.
template	A template is HTML code associated with a web page. It defines the layout of the page and also where to get HTML for each part of the page. In PeopleSoft Enterprise, you use templates to build a page by combining HTML from a number of sources. For a PeopleSoft Enterprise portal, all templates must be registered in the portal registry, and each content reference must be assigned a template.
territory	In PeopleSoft Enterprise Sales Incentive Management, hierarchical relationships of business objects, including regions, products, customers, industries, and participants.
third party	A company or vendor that has extensive PeopleSoft Enterprise product knowledge and whose products and integrations have been certified and are compatible with PeopleSoft Enterprise applications.
3C engine	Abbreviation for <i>Communications, Checklists, and Comments engine</i> . In PeopleSoft Enterprise Campus Solutions, the 3C engine enables you to automate business processes that involve additions, deletions, and updates to communications, checklists, and comments. You define events and triggers to engage the engine, which runs the mass change and processes the 3C records (for individuals or organizations) immediately and automatically from within business processes.
3C group	Abbreviation for <i>Communications, Checklists, and Comments group</i> . In PeopleSoft Enterprise Campus Solutions, a method of assigning or restricting access privileges. A 3C group enables you to group specific communication categories, checklist codes, and comment categories. You can then assign the group inquiry-only access or update access, as appropriate.
TimeSpan	A relative period, such as year-to-date or current period, that can be used in various PeopleSoft Enterprise General Ledger functions and reports when a rolling time frame, rather than a specific date, is required. TimeSpans can also be used with flexible formulas in PeopleSoft Enterprise Projects.

trace usage	In PeopleSoft Enterprise Manufacturing, enables the control of which components will be traced during the manufacturing process. Serial- and lot-controlled components can be traced. This is maintained in the Item Master record.
transaction allocation	In PeopleSoft Enterprise Incentive Management, the process of identifying the owner of a transaction. When a raw transaction from a batch is allocated to a plan context, the transaction is duplicated in the PeopleSoft Enterprise Incentive Management transaction tables.
transaction state	In PeopleSoft Enterprise Incentive Management, a value assigned by an incentive rule to a transaction. Transaction states enable sections to process only transactions that are at a specific stage in system processing. After being successfully processed, transactions may be promoted to the next transaction state and “picked up” by a different section for further processing.
Translate table	A system edit table that stores codes and translate values for the miscellaneous fields in the database that do not warrant individual edit tables of their own.
tree	The graphical hierarchy in PeopleSoft Enterprise systems that displays the relationship between all accounting units (for example, corporate divisions, projects, reporting groups, account numbers) and determines roll-up hierarchies.
tuition lock	In PeopleSoft Enterprise Campus Solutions, a feature in the Tuition Calculation process that enables you to specify a point in a term after which students are charged a minimum (or <i>locked</i>) fee amount. Students are charged the locked fee amount even if they later drop classes and take less than the normal load level for that tuition charge.
unclaimed transaction	In PeopleSoft Enterprise Incentive Management, a transaction that is not claimed by a node or participant after the allocation process has completed, usually due to missing or incomplete data. Unclaimed transactions may be manually assigned to the appropriate node or participant by a compensation administrator.
universal navigation header	Every PeopleSoft Enterprise portal includes the universal navigation header, intended to appear at the top of every page as long as the user is signed on to the portal. In addition to providing access to the standard navigation buttons (like Home, Favorites, and signoff) the universal navigation header can also display a welcome message for each user.
update access	In PeopleSoft Enterprise Campus Solutions, a type of security access that permits the user to edit and update data. See also <i>inquiry access</i> .
user interaction object	In PeopleSoft Enterprise Sales Incentive Management, used to define the reporting components and reports that a participant can access in his or her context. All PeopleSoft Enterprise Sales Incentive Management user interface objects and reports are registered as user interaction objects. User interaction objects can be linked to a compensation structure node through a compensation relationship object (individually or as groups).
variable	In PeopleSoft Enterprise Sales Incentive Management, the intermediate results of calculations. Variables hold the calculation results and are then inputs to other calculations. Variables can be plan variables that persist beyond the run of an engine or local variables that exist only during the processing of a section.
VAT exception	Abbreviation for <i>value-added tax exception</i> . A temporary or permanent exemption from paying VAT that is granted to an organization. This terms refers to both VAT exoneration and VAT suspension.
VAT exempt	Abbreviation for <i>value-added tax exempt</i> . Describes goods and services that are not subject to VAT. Organizations that supply exempt goods or services are unable to recover the related input VAT. This is also referred to as exempt without recovery.

VAT exoneration	Abbreviation for <i>value-added tax exoneration</i> . An organization that has been granted a permanent exemption from paying VAT due to the nature of that organization.
VAT suspension	Abbreviation for <i>value-added tax suspension</i> . An organization that has been granted a temporary exemption from paying VAT.
warehouse	A PeopleSoft Enterprise data warehouse that consists of predefined ETL maps, data warehouse tools, and DataMart definitions.
work order	In PeopleSoft Enterprise Services Procurement, enables an enterprise to create resource-based and deliverable-based transactions that specify the basic terms and conditions for hiring a specific service provider. When a service provider is hired, the service provider logs time or progress against the work order.
worker	A person who is part of the workforce; an employee or a contingent worker.
workset	A group of people and organizations that are linked together as a set. You can use worksets to simultaneously retrieve the data for a group of people and organizations and work with the information on a single page.
worksheet	A way of presenting data through a PeopleSoft Enterprise Business Analysis Modeler interface that enables users to do in-depth analysis using pivoting tables, charts, notes, and history information.
worklist	The automated to-do list that PeopleSoft Workflow creates. From the worklist, you can directly access the pages you need to perform the next action, and then return to the worklist for another item.
XML link	The XML Linking language enables you to insert elements into XML documents to create a links between resources.
XML schema	An XML definition that standardizes the representation of application messages, component interfaces, or business interlinks.
XPI	Abbreviation for <i>eXtended Process Integrator</i> . PeopleSoft XPI is the integration infrastructure that enables both real-time and batch communication with JD Edwards EnterpriseOne applications.
yield by operation	In PeopleSoft Enterprise Manufacturing, the ability to plan the loss of a manufactured item on an operation-by-operation basis.
zero-rated VAT	Abbreviation for <i>zero-rated value-added tax</i> . A VAT transaction with a VAT code that has a tax percent of zero. Used to track taxable VAT activity where no actual VAT amount is charged. Organizations that supply zero-rated goods and services can still recover the related input VAT. This is also referred to as exempt with recovery.

Index

A

- abs() 433
- acos() 434
- addCtrl() 354
- addDays() 434, 439
- Additional Attribute file 396
- additional documentation xxvi
- addMonths() 434, 439
- addYears() 434, 439
- Administration console
 - description 415
 - understanding 3
- administration, mobile solutions 395
- ALL selection option 99
- Allow options 103
- and() 437
- annotation, Package 408
- ANY selection option 99, 100
- APIs
 - calling the compound model 379
 - client operations 155
- application fundamentals xxv
- application page example 352
- application root, mobile configuration 404
- application server 4, 297
 - default port 7777 308
 - Java and CLASS files 311
 - path to JSP pages 302
- Application Why Help 460
- architecture
 - compound model 126, 376
 - understanding 4
- asin() 434
- ATTR element 212
- Attribute element 262
- Attribute field
 - Display page, internal solution 283
 - Domain Member Min/Max dialog 113
- Attribute option, Price page 286
- attribute parameter, getControlData 166
- attribute record 316
- Attribute to Display field 67
- attributes
 - assigning date attributes 74

- assigning values to attributes 74
- attaching metadata to selection points 118
- classes 13
- for domain member quantities 114, 116
- for external data 396
- form control 326
- in modeling 12
- obtaining values through the API 165
- parameter, getControlData 166
- warning about number of characters 119
- Attributes field
 - compound model 137
 - group control insertion 367
 - list control insertion 365
 - table control insertion 368
- Auto Submission of Picks option 67
- avgWithQty() 434

B

- batch configuration 265, 266
- binding, selection points 31
- bnd() 434, 437, 441
- bnd() 31
- bnd() function 25
- boolean comparison 21
- boolean functions 436
- business logic 4
- button controls
 - editing 374
 - inserting 363
- Button Type option
 - button control insertion 363

C

- cache size 427
- cached resources 164
- calico.page.restore 349
- calico.page.start 349
- calico.solution.description 349
- calico.solution.name 349
- calico.solution.restorePolicy 349
- calico.solution.version 349
- CalicoConstantsInc.jsp 321

- CalicoControlInc.jsp 322
- CalicoEndFormInc.jsp 322
- CalicoNA.properties 304
- CalicoProcessForm.jsp 320
- CalicoProcessFormInc.jsp 316, 321
- CalicoStartFormInc.jsp 313, 321
- CalicoUI.properties 361
- Cancel order button 421
- Caption field 283
- Caption Type option
 - Application Why Help 372
 - group control insertion 366
 - Insert Numeric Data dialog box 373
 - list control insertion 364
 - table control insertion 368
- character limitations, attribute values 117
- Choice class 158, 180
- Choice object 317
- choices 155
- CHOICES element 197, 203, 242
- Choices field 138
- Class (CL) attribute 218
- CLASS files 311
- class, custom, initializing the WCP
 - with 314
- classes
 - adding attributes to 72
 - creating 69
 - date attributes 74
 - deleting 70
 - modeling 13
 - move, copy, and paste 71
 - naming restrictions 70
- classpath of JSP servlet 309
- Client Operations Processor (COP)
 - API, description 155
 - initializing 162
 - making calls to 316
 - runtime processing 313
 - web application 301
- ClientOperations class
 - methods 161
 - summary 157
- .cms files 58
- Column Heading field 368
- COM messaging 402
- command line compile 430
- comments, submitting xxx
- common elements xxx
- Comparator object 175
- compareTo() 434, 437, 439
- comparison constraints
 - description 21
 - editing 104
- compatibility constraints 98
 - See Also* editing
 - overview 15
 - vs. requirement constraint 46
- compilation of model 65
- compile ID 65, 162
- Compile ID option 66
- completeness information 259
- component models
 - adding or removing 135
 - configuring in batch mode 265
 - delta configuration information 233
- component, configurable, *See* configurable component
- COMPONENT_DEFINITION
 - element 261
- Components and Files View 56
- COMPONENTS element 237
- Components field 289
- compound models
 - architecture 126
 - compiling, running, testing 146
 - configuring in batch mode 266
 - connections 127
 - creating a project 130
 - delta configuration information 234
 - deploying on a web application 303
 - properties file 463
 - relationships 127
 - sample 46, 467
 - structure types 123
 - uses for compound 123
 - xml, compound structure definition
 - sample 468
- Compound Violations field 289
- compression of data 428
- concatenate() 434
- CONFIG_DETAILS element 232
- configCopy attribute 263
- configId attribute 263, 269
- configName attribute 263
- configurable components
 - creating 132
 - deleting 134
 - editing types 147
 - rearranging 135

- configuration
 - attributes 169
 - data 158
 - loading, API call 162
 - mobile environment 387
 - state 298
 - Configuration Attributes field 289
 - Configuration class
 - methods 169
 - summary 158
 - configuration details 187, 289
 - Configuration Details option 288
 - CONFIGURATION element 231, 260
 - Configuration HTML Page 291
 - configuration information
 - retrieving 201
 - configuration list price 286
 - configuration records 298
 - information in 315
 - saving in external database 317
 - configuration results 292
 - configuration session, ending 164
 - Configuration Type option 281
 - configurations 201
 - See Also* configuration information
 - batch mode 265, 266
 - copying 263
 - restore policy 303
 - retrieving stored 267
 - save and restore 170
 - saving 229
 - saving using COPXML request 267
 - updating 186, 195
 - verifying 169
 - Configurator Form 360
 - Configurator Solution Tester 278
 - Configuring button 419
 - conflicted state 173
 - Connected Components field 139
 - connecting to CRM applications 275
 - connection points
 - creating and editing types 150
 - description 127
 - connections 467
 - CONNECTIONS element 258
 - Connections field, Request Details
 - page 289
 - constraints
 - bound and unbound 26
 - comparison 21
 - compatibility 15
 - directional, non-compatibility 16
 - dynamic default 19
 - elimination 21
 - non-compatibility 16
 - requirement 18
 - resource 20
 - summation 20, 21
 - contact information xxx
 - contains() 437, 441
 - control data, retrieving 158
 - Control Size field
 - Display page, internal solution 283
 - list control insertion 364
 - Control Type option 283
 - ControlData class
 - methods 171
 - summary 158
 - ControlData object
 - COP API 157
 - retrieving 165
 - ControlItem class
 - description 176
 - summary 158
 - ControlItem object 157
 - COP, *See* Client Operations Processor
 - COPXML servlet 185
 - COPXML servlet statistics 430
 - Copy button 419
 - cos() 434
 - cot() 434
 - countWithQty() 434, 437
 - creating "None" 73
 - cross-references xxix
 - .csp files 58
 - .csw files 58
 - Current Version 61
 - Customer Connection website xxvi
 - customization, Mobile Configurator
 - client 398
- D**
- data 317, 389
 - compressing configuration data 428
 - configuration 389
 - configuration deltas 383
 - domain members 204
 - dynamic default quantity storage 80
 - loading form control data from the model 354

- model 389
- model, internalizing with Administration
 - console 396
 - numeric form control 339
 - numeric object 372
- database abstraction 4
- databases
 - connecting Visual Modeler to 54
 - interface setup 51
 - specifying a default in Visual Modeler 54
 - supported 47
- date() 434, 439
- dates
 - comparison constraint 21
 - constants in expressions 27
 - effectivity 24, 25, 30
 - functions 438
- dateToInt() 434, 439
- daysBetween() 435, 439
- DB Table format 30
- DB2, *See* IBM DB2
- Debug option
 - Configurator Installation page 278
 - Solution Tester 422
- decision points 156, 192
 - See Also* selection points
 - and selection points 325
 - choices 156
 - description 157
 - retrieving domain members of 204, 206
 - states 173
- DECISION_POINTS element 192
- Def Choice option, domain member
 - quantity dialog 84
- default quantities 34, 35
- default quantities, static 43
- Default Value field, Extern Manager 92
- defaults
 - choices and quantities 115
 - compound models 137
 - getting selections through attributes 116
- Defaults column 81
- Defaults field 149
- Define Request link 288
- Define Stylesheet link 288
- Delete order button 421
- delimiter token 80
- delta information
 - DELTA_INFO element 233
 - displaying in custom UI 350
- Delta Information option 289
- delta price (DPR) attribute 217
- Delta Price Only option 286
- delta pricing
 - displaying in custom Ui 351
 - specifying through the API 168
- DELTA_INFO element 260
- delta-pricing information 164
- deltas, configuration 383
- deployment
 - Administration console 415, 416
 - custom UI 297, 302, 303, 304
 - mobile configuration environment 395
- directional compatibility 15
- directional compatibility constraints 46
- directory structure, custom UI files 311
- Display Component Violations field 289
- display information
 - API call 172
 - ControlData class 158
 - ControlItem class 158
 - domain member 176
 - getting through the API 169
- display options 283
- Display page 279
 - External Solution 280
 - Internal Solution 281
- display properties
 - CalicoUI.properties 354
 - specifying 350
- Display Selection Violations field 289
- DM Min/Max field 83
- DMChoice class 159, 180
- documentation
 - printed xxvi
 - related xxvi
 - updates xxvi
- doesNotContain() 437, 441
- doesNotEqual(441
- doesNotEqual() 435, 437, 439
- domain members
 - creating a “None” 73
 - creating internal 73
 - date attributes 74
 - deleted 176
 - description 14, 156
 - external to model 14

- filtering 182, 209
- internal to model 14
- min/max limits 41
- retrieving the display information
 - of 176
- retrieving values of all 213
- retrieving values of selected 215
- retrieving the state 177
- retrieving through COP API 204
- setting quantity limits 112
- setting up binding for external 75
- sorting 207
- state flags 178
- Dreamweaver, *See* MacroMedia Dreamweaver
- Dreamweaver extensions, using 359
- dynamic default relationships
 - default quantities 108
 - example 98
 - storing the quantity in a database 80
- dynamic defaults
 - editing 100
 - overview 19
- dynamic presentation 4

E

- effectivity dates
 - constraints 30
 - description 24
- Effectivity Settings field 62
- eliminated items 324
- elimination constraints
 - description 21
 - editing 103
- Enabled property 29
- endsWith() 437, 441
- equals() 435, 438
- error messages 191
- error-checking on expressions 89
- errors in custom UI creation 356
- evaluation of expressions 31
- EVChoice class 159, 181
- Events field
 - group control insertion 367
 - list control insertion 364
 - table control insertion 368
- example of configuration 291
- Exclusive property 29
- Explanation field
 - Domain Member Min/Max dialog 113

- domain member quantity dialog 83
- elimination editor 103
 - SP Min/Max 83
- explanations
 - creating parameterized 44
 - incomplete configuration vs. minimum violation 43
 - relationships 27
 - returning using COPXML 221
- EXPLANATIONS attribute 209, 221
- Explanations.properties
 - copying 429
 - description of use 429
 - searching 429
- export, model
 - e 63
- Expression editor 90
- Expression field
 - Domain Member Min/Max dialog 113
 - Expression editor 89
 - Price page 286
- expression values, retrieving from a database 81
- expressions
 - adding to the RHS of a relationship 96
 - behavior at run time 26
 - calculating the required quantities 32
 - creating 91
 - creating and editing, overview 89
 - creating relationships containing 94
 - date functions 438
 - default values 31
 - deleting 91, 96
 - format in LHS 81
 - functions and operators 433
 - in relationships 25
 - string functions 441
 - viewing 96
- Expressions field, Request Details page 289
- extern entry
 - template 340
 - text box control 462
- External Choices field 138
- external data
 - making internal to the model 85
 - parameterized explanations 44
 - variables 91
- external database, saving configuration records to 317

- external domain members 75
- external node 277
- External Node field 278
- External Solution option 280
- external solutions, schemas 280
- external variables 91, 253
- External Variables field, Request Details page 289
- externs
 - API 159
 - binding 31
 - creating 91
 - format in RHS 81
 - representing in the API 159
 - retrieving by COPXML request 250
 - retrieving from a database 81
- ExternVar class
 - methods 183
 - summary 159

F

- Field Processing option 283
- Filename field 81
- filter parameter 167
- filtering
 - API 159
 - domain members 166, 182
 - methods 175
 - table data for the model 80
- Find window 57
- First Item Text field 364
- FLAG element 255, 261
- flag, internal 13
- FLAG_SET element 255, 262
- flags, state of domain members 178
- font styles in Properties table 74
- form control data, loading 354
- form control inclusion set 343
- form control item 300
- form control templates
 - application why help 337
 - customizing 353
 - description 300
 - extern entry 340
 - file location 342
 - form control why help 338
 - implementing 341
 - modifying, example 355
 - multi-select group 327
 - multi-select list 329

- multi-select table 331
- numeric data 339
- registering modified templates 354
- single-select group 327
- single-select image 332
- single-select image table 336
- single-select list 328
- single-select table 330
- text input 338
- understanding 325
- Form Control Why Help 460
- form controls
 - list of provided 299
 - plugging into pages 342
 - processing 319, 320
- format checking, externs 92
- Format property 29, 98
- forms
 - editing 374
 - inserting 362
- Frame Dimensions fields
 - Display page, external solution 280
 - Display page, internal solution 281
- Function Category field 89
- Function Name field 90
- functions
 - for expressions 433
 - user-defined 91

G

- getAttributeNames() method 173
- getAttributes 177
- getAttributeValue(String name) method 173
- getAttributeValues method 173
- getBeginningOfMonth() 439
- getBeginningOfWeek() 439
- getBeginningOfYear() 440
- getClientOperations 313, 316
- getCompileVersion 317
- getConfigurationRecords 313, 317
- getDay() 435, 440
- getDeltaPrice 177
- getExternVar 169
- getFlags 177
- getMaxQty 177
- getMinQty 177
- getModelCompileVersion 313
- getModelName 313, 317
- getModelVersion 313, 317

getMonth() 435, 440
 getNumericData 169
 getObjectNames 313
 getQty 177
 getQty method 172
 getQuantity method 450
 getQuantity() 448
 getSolveDate() 440
 getState method 172
 getting stored configuration records 317
 getToday() 440
 getTotalPrice 169
 getViolations 169, 177
 getViolations method 172
 getYear() 435, 440
 glossary 513
 group controls, inserting 365

H

hasEliminationLevel 177
 hierarchy, model files 65

I

IBM DB2 51
 IBM DB2 database setup 52
 ID field
 group control insertion 366
 list control insertion 364
 table control insertion 368
 if() 435
 iFrame Height field, Solution Tester 422
 image controls, inserting 368
 import, model 64
 Include directive, JSP 343
 include() 343
 inclusion set
 parameters 344
 using for form control templates 343
 inclusion set for form control 343
 Incomplete Configuration Explanation
 field 62
 indexOf() 435, 441
 initial page display, API cal 164
 initialize 313
 Insertable Objects dialog box 90
 Installation Table page 277
 Instance field
 connection point 141
 connection point type 152

int getMaxChoices method 173
 int getMinChoices method 173
 integration
 getting pricing data 168
 implementing 6
 Mobile Configurator 402
 Mobile Configurator and ERP
 systems 400
 setting up 276
 to PeopleSoft Enterprise CRM 275
 with Order Capture 273, 275
 Integration Broker, setup 277
 internal flag 13
 Internal format 29
 internal model data 67
 internal node 277
 Internal Node field 278
 Internal property 13
 internal solutions, schemas 281
 intToDate() 435, 440
 Invalid Type Explanation, Extern
 Manager 93
 isValid r 177
 ItemFilter class
 description 182
 summary 159
 ItemIterator class
 methods 183
 summary 159
 iteration 159

J

Java Server Pages (JSP)
 basis applications pages 311
 runtime processing 309
 scripting in runtime processing 310
 Java source files 311
 JavaServer Pages (JSP) 307
 compound model 377
 compound models 126
 description 300
 implementing in a solution 322
 JNDIDBName.properties 53
 JSP, *See* Java Server Pages (JSP)

L

Label field 363
 length() 435
 Levels format 30

- List Configurations button 420
- list control 363
- List Price Source field 285
- loadConfigurationRecords 313, 317
- localization
 - Mobile Configurator client 399
 - solution UI 398
 - specifying properties 347

M

- MacroMedia Dreamweaver 47
- maintenance
 - mobile packages 393
 - model 116
 - reducing for model 45
- major and minor versions 65
- Major Version field 61
- Major Version, Minor Version fields,
 - compound model settings 131
- Manufactured Components option 288
- Max Occurs field 133
- max() 435
- Maximum field, domain member quantity
 - dialog 83
- Maximum Number 83
- Maximum Occurrences field 149
- maxWithQty() 435
- memory use 427
- message display 27
- messaging node
 - associating to CRM applications 277
- metadata, on selection points 118
- Min Occurs field 133
- Min of, Max of quantity policies 82, 115, 118
- min() 435
- Min/Max settings
 - interaction with default quantities 42
 - selection points and domain members 40, 41
- Minimum field, domain member quantity
 - dialog 83
- Minimum Number field 82
- Minimum Occurrences field 149
- Minor Version field 61
- minWithQty() 435
- Mobile Configurator COM API 471
- Mobile Transfer Service APIs 400
- Mode option 285
- model data 67

- Model field
 - component model editor 132
 - defining component type 148
- Model ID field 282
- model information 186, 189
- Model Information File field 362
- Model Name and Model Version Number
 - field 362
- Model Structure View 55
- Model Tester 47, 424
 - behavior, backing out of a pick sequence 43
 - compiling a model 66
 - expression behavior 26
 - using 66
- Model Tester link
 - Display page, external solution 280
 - Display page, internal solution 282
- Model Version field
 - Model Tester 66
- modeling
 - attributes 12
 - classes 13
 - concepts 11
 - objects 12
 - quantities in 32
 - summary of process 46
 - team environment 146
- models
 - compiling 65
 - compiling from command line 430
 - exporting and importing 63
 - loading 426
 - managing versions 425
 - sample 46
 - sample compound model 381
 - uses for compound 123
 - versioning 65
- Multi-Sel option, selection point
 - properties 82
- multi-select controls
 - number of selections allowed 41
 - overview 360
 - specifying 111
- multi-select group 327
 - JSP code 454
- multi-select list
 - JSP code 456
 - state tags 329
 - template 329

- multi-select objects
 - directional compatibility 15
 - directional compatibility constraint 46
- multi-select table control
 - JSP code 457
 - template 331

N

- “Name” attribute 107
- naming restrictions
 - class names 70
 - domain members 73
 - queried data 77
- New button 418
- newState attribute 269
- non-compatibility constraints 16
- non-directional compatibility 15
- None domain member 73, 77
- None field 284
- none option
 - in dynamic defaults 100
 - providing through the API 171
- not() 438
- notes xxix
- Number field 113
- Number of Columns field 283
- Number of Controls Per Row field 67
- numeric comparison 21
- numeric data control 339, 461
- Numeric Data field
 - connection point 141
 - connection point type 152
- numeric data object 169, 372
- numeric values, component 252
- NUMERIC_VALUES element 223
- NumericData class
 - methods 183
 - summary 160

O

- objectName parameter 166
- occursAfter() 438, 440
- occursBefore() 440
- occursOnOrAfter() 438, 440
- occursOnOrBefore() 438, 440
- ODBC data source configuration 51, 52
- On Output property 119
- Operation field
 - connection point type 151

- Relationship editor 140
- Operator option 286
- operators, numeric 433
- Optional option, domain member quantity dialog 83
- or() 438
- Oracle database setup 51, 52
- order change 269
- orderChange attribute 269
- Other UI field 62
- out object 311
- output 287, 422
- Output page 279
- overridable quantity policy 82, 115
- Overview window 57

P

- Package Components option 288
- Packaging Tool 390
- page flow in custom UI 319
- page template 300
- parameters
 - explanations 44
 - form control 326
 - format in explanation 27
- PCIF
 - list of elements 477
 - use 63
- PeopleBooks
 - ordering xxvi
- PeopleCode, typographical conventions xxviii
- PeopleSoft Advanced Configurator 275
 - access points 290
 - integrating 290
 - integrating, overview 290
 - launching 290
 - testing and administration 3
- PeopleSoft Configurator Server Location field 132
- PeopleSoft Enterprise Visual Modeler 11
- PeopleSoft.MCfkgProxy 471
- performance
 - enhancing at model level 45
 - performance optimization 302
- pi() 435
- plus-minus pricing 283
- policy options 163
- Port field 61
- prerequisites xxv

- Price page 279, 284
- pricing
 - establishing options 284
 - properties 285
- pricing information
 - getting through the API 168
 - setting in custom UI solutions 324
- print command, in scripting JSP pages 311
- printed documentation xxvi
- processConfigurationRecords() 313
- Product Selector option 288
- product() 435
- project creation 59
- project files 58
- project settings
 - compound model 131
 - specifying 60
- properties
 - class 13
 - compound models 304
 - displaying custom UI state tags 350
 - form control 326
 - loading for a custom control 354
 - mobile solution packages 408
 - of relationships 29
 - specifying for solution's models 347
 - specifying solution information 349
- Properties Editor
 - description 56
 - font styles 74
- PSMSGNODEDEFN table 278
- Purchased Components option 288

Q

- quantities
 - default, interaction with Min/Max settings 42
 - domain member min/max limits 41
 - dynamic default 34
 - getting from attributes 116
 - of domain members, retrieving 219
 - running a check 116
 - selection point 43
 - setting default values 115
 - setting up in model 109
 - setting default 114
 - static default 34
- Quantity option 81
- quantity policies 82

- Quantity Policy
 - description 42
 - setting 117
- Quantity Policy column, Edit Default Choices dialog 115
- Quantity Value/Expression column
 - Edit Default Choices dialog 115
- quotient() 436

R

- Recurring Price Source field 286
- recurring prices 286
- Reference field, component model editor 132
- Refresh Functions From Server button 90
- related documentation xxvi
- relationships
 - between objects 14
 - See Also* constraints
 - compound models 127
 - creating 93
 - creating and deleting in compound models 138
 - creating outside the model 105, 107
 - deleting 91, 96
 - displaying for compound model 141
 - dynamic default with quantity 117
 - explanations 27
 - expressions on the LHS 98
 - prerequisites 87
 - sample compound model 382
 - specifying required, in compound model 142
 - viewing 96
 - working with 96
- release 313, 318
- Render With option 288
- reparent classes 71
- Replace "None" With field
 - group control insertion 367
 - list control insertion 364
 - table control insertion 368
- Request Details page 287
- request details, internal solutions 288
- Request Message field 288
- request properties 288
- Required option, Extern Manager 92
- Required Relationship field 133
- requirement constraints
 - description 18

- editing 99
- externs 92
 - vs. compatibility constraint 46
- requirements, Mobile Configurator 393
- reserved symbols and words 13
- reset buttons 363
- resetConfiguration 313, 318
- resource constraints
 - description 20
 - editing 101
- restore API call 162
- Restore button 419
- restore configuration 158, 317
- restore policy 303, 397
- Restore Policy field
 - component model editor 132
 - defining component type 149
- Restore Policy option
 - Display page, internal solution 283
- restricted characters 132
- Return field 284
- Return Type option 89
- RHS, *See* right-hand-side of a relationship
- root class 13
- round() 436
- Routing Operations option 289
- runtime functions 375
- runtime objects 359, 362

S

- sample configuration 291
- samples
 - compound model 383, 467
 - creating a custom UI 380
 - custom application page 352
 - Mobile Configurator 401
- Save order button 421
- Schema ID 280, 281
- schemas
 - creating for external solutions 280
 - creating for internal solutions 281
 - display 279
 - external solution 279
 - internal solution 280
 - output 279
 - pricing 279
 - understanding 278, 279
- scriptlets 310
- SECTION element 235, 261
- selectable state 173
- selected state 173
- Selection Point option
 - Application Why Help 372
 - Dreamweaver setup 373
 - group control insertion 366
 - list control insertion 364
 - table control insertion 368
- selection points
 - deleting 84
 - described 81
 - description 14
 - form controls for 360
 - min/max limits 40
 - multi-select 220
 - setting visible attributes 84
 - specifying quantities 32
 - viewing 84
- SELECTION_POINT element 262
- selections
 - number allowed 40, 42, 43, 110
 - setting default 114
 - setting defaults 115
- Selections field, Request Details page 289
- Separator Settings field 62
- Sequence field 283
- Server field 61
- server node 277
- server, application 297
- servlet statistics 430
- servlet, custom UI 309
- setPricingControls() 351
- Show Application Violations, Display page, internal solution 284
- Show Configuration List Price option, Display page, internal solution 284
- Show Delta Price option 283
- Show Eliminated option
 - Display page, internal solution 283
 - group control insertion 366
 - list control insertion 364
 - table control insertion 368
- Show Elimination Level option 31, 67
- Show None on Required Controls option 67
- Show "None" option
 - group control insertion 366
 - list control insertion 364
 - table control insertion 368
- Show Violations option
 - Display page, internal solution 283

- sin() 436
- single-select controls
 - number of selections allowed 41
 - overview 360
 - specifying 111
- single-select group control
 - JSP code 453
 - template 327
- single-select image control
 - Dreamweaver extension 360
 - JSP code 458
 - template 332
- single-select image table control
 - description 336
 - JSP code 458
- single-select list
 - state tags 328
 - template 328
- single-select list control
 - JSP code 456
- single-select table
 - JSP code 455
 - template 330
- site support 389
- Solaris, form control location 342
- solution information 349
- solution list 349
- solution package
 - adding a model 407
 - adding files 408
 - contents 388
 - creating 405
 - generating and updating the installer 408, 409
 - modifying 408
 - printing contents 410
 - updating the model 409
- Solution Package 303
- solution properties 280
- solution state 158
- Solution Tester
 - description 418
 - launching 421
- Solution Tester link
 - Display page, external solution 280
 - Display page, internal solution 282
- Solution Type option
 - Display page, external solution 280
 - Display page, internal solution 281
- solutionId attribute 263, 269
- solutions
 - creating with Dreamweaver 361
 - distribution and installation for mobile use 389
 - external vs. internal 279
 - integrated 6
 - setting test parameters 422
 - specifying model properties 347
 - standalone 6
 - testing 418
- solve date 25
- Solve Date (YYYYMMDD) field 67
- Sort button 366
- Sort By State option 67
- Sort option
 - list control insertion 364
 - table control insertion 368
- sort parameter, getControlData 166
- sorting
 - custom 175
 - domain members 166, 207
 - methods 174
- sortsAfter() 438, 442
- sortsBefore() 438, 442
- source control
 - Mobile Configurator 402
 - model files 62
 - setup 50
 - supported software 47
 - Visual Modeler 46
- Source field
 - connection point 141
 - connection point type 151
- SP Min/Max button, selection point properties 82
- SQL queries
 - creating relationships outside the model 105, 107
 - writing 75, 76, 77, 78, 79
- SQL Query
 - domain member quantity dialog 83
 - field 72
- SQL Query edit window 83
- SQL Query format 29
- SQL Query option 13
- SQL Server setup 51
- sqrt() 436
- standalone solution
 - implementing 6
- start page, setting 395

startsWith() 438, 442
 state
 decision point, getting 173
 domain member 177
 of domain members, retrieving 219
 state tags 328, 329, 350
 state, model, clearing 318
 static default quantities 43
 static variable 91
 string functions 441
 String or Path field
 group control insertion 366
 list control insertion 364
 table control insertion 368
 String or Path option 374
 StringBuffer.append() 357
 STRUCTURE element 261
 Structure field 289
 STRUCTURE, SUBSTRUCTURE
 elements 256
 stylesheet for internal solution UI 288
 Submit order button 421
 substring() 436, 442
 SUBSTRUCTURE element 261
 suggestions, submitting xxx
 sum quantity policy 82
 Sum quantity policy 115, 118
 sum() 436
 summation relationships
 description 20
 dialog box 94
 editing 102
 sumWithQty() 436

T

Tab field, Display page, internal
 solution 283
 table aliases 79
 table controls, inserting 367
 table data 80
 tan() 436
 Target DP field 141
 Target field
 compound model relationship 139
 connection point 141
 connection point type 152
 Target Variable field 151
 team modeling 146
 TelcoSample 380
 template

 form control 300
 page 300
 templates, *See* form controls
 compound models 128
 extern entry form control 340
 form control filenames 341
 form controls 353
 implementing for form controls 341
 terms 513
 Test UI field 61
 testing and administration tools 3
 text input control 461
 text input template 338
 toDate() 440
 toDegrees() 436
 toFloat() 436, 438, 440
 toInteger() 436, 438, 440
 toLowerCase() 442
 toRadians() 436
 TOTAL_PRICE attribute 203, 236
 toUpperCase() 442
 transfer service, Mobile Configurator 401
 trim() 442
 Type field
 connection point 140
 Type field, component model editor 132
 Type field, Extern Manager 92
 Type option
 Display page, internal solution 283
 Price page 286
 typographical conventions xxviii

U

UI Version Number field 362
 un-installing
 mobile solution package 393
 Unbounded button, Domain Member
 Min/Max dialog 113
 undefined state 173
 Update field
 Display page, internal solution 284
 URL field 363
 Use Min/Max option 81
 Use Most Current Version option 282
 Use Quantity Policy for All Domain
 Members option 115
 Use Select (List) Controls option 67
 user-defined functions
 creating and adding 443
 description 91

- getQuantity() 448
- retrieving 90
- UserFunction interface 444
- UserFunctions.xml 446

V

- Validate on Return option 284
- Value (VL) attribute 224
- variable
 - extern 91
 - external 91
 - static 91
- verification
 - of configuration 169
 - quantity calculation of domain members 116
- Verify Configuration option 67
- version control, *See* source control
- versioning, models 65
- versions
 - assigning to model 348
 - editing component model's in a compound 144
 - for component models of a compound 136
 - getting the compile 317
 - mobile solutions 393
 - of models 425
 - retrieving the latest compile 189, 190
- View Details – XML button 420
- View Details button 420
- Violation class 184
- violation messages 429
- violations
 - API 160
 - externs 92
 - incomplete configuration vs. minimum violation explanations 43
 - returned by COPXML 250
 - returning choice violations 249
 - returning explanations of constraint violations 210
- VIOLATIONS element 236
- visual cues xxix
- Visual Modeler
 - description 11
 - main window 55
- Visual SourceSafe 62

W

- warnings xxix
- Web Client Processor (WCP)
 - architecture component 301
 - initializing 314
 - methods 317
 - releasing 318
 - runtime processing 313
- web components 300
- web deployment 297
- web server
 - architecture 4
 - default port 80 308
- Why Help
 - application-level 460
 - constraint violation explanations 27
 - externs 92
 - form control template 338
 - inserting 371
 - template for application-level 337
- workspace creation 59

X

- xml, compound model structure
 - definition 468
- xor() 438