



PeopleTools 8.12 Integration Tools PeopleBook

SKU MTITr8SP1B 1200

PeopleBooks Contributors: Teams from PeopleSoft Product Documentation and Development.

Copyright © 2001 by PeopleSoft, Inc. All rights reserved.

Printed in the United States of America.

All material contained in this documentation is proprietary and confidential to PeopleSoft, Inc. and is protected by copyright laws. No part of this documentation may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, including, but not limited to, electronic, graphic, mechanical, photocopying, recording, or otherwise without the prior written permission of PeopleSoft, Inc.

This documentation is subject to change without notice, and PeopleSoft, Inc. does not warrant that the material contained in this documentation is free of errors. Any errors found in this document should be reported to PeopleSoft, Inc. in writing.

The copyrighted software that accompanies this documentation is licensed for use only in strict accordance with the applicable license agreement which should be read carefully as it governs the terms of use of the software and this documentation, including the disclosure thereof.

PeopleSoft, the PeopleSoft logo, PeopleTools, PS/nVision, PeopleCode, PeopleBooks, Vantive, and Vantive Enterprise are registered trademarks, and *PeopleTalk* and "People power the internet." are trademarks of PeopleSoft, Inc. All other company and product names may be trademarks of their respective owners.

Contents

About This PeopleBook

Before You Begin	xvi
Related Documentation	xvi
Documentation on the Internet	xvi
Documentation on CD-ROM	xvii
Hardcopy Documentation	xvii
Typographical Conventions and Visual Cues	xvii
Comments and Suggestions	xix

Chapter 1

Overview of Integration Technologies

Application Messaging	1-1
When to Use Application Messaging	1-1
Component Interfaces	1-2
When to Use Component Interfaces	1-2
Business Interlink	1-2
When to Use Business Interlinks	1-3
Integration Software Development Kit	1-3
Related Documentation	1-4
File Layout Objects / Definitions	1-5
When To Use File Layouts	1-5
EDI Manager	1-5
When To Use EDI Manager	1-6
Merchant Integration	1-6
When To Use Merchant Integration	1-7
Message Agent	1-7
When To Use Message Agent	1-7
Database Agents	1-7
When To Use Database Agents	1-8

Chapter 2

Merchant Integration

Understanding Merchant Integration	2-1
--	-----

Merchant Integration Process Flow.....	2-1
The User Experience	2-2
Hidden Application Functionality	2-2
Branded Application Functionality	2-3
Merchant HTML Content	2-3
MIP Features	2-3
Maintaining Merchant Information	2-5
Merchant Categories Component.....	2-5
Merchant Categories Page.....	2-6
Merchant Profile Component.....	2-7
Merchant Profile Page.....	2-7
Merchant Authentication Page.....	2-9
Merchant BI Overrides Page.....	2-10
Merchant Category Page.....	2-12
Application Attributes Page	2-13
Developing an MIP.....	2-14
Single Sign-On Records and Fields	2-15
PSAUTHPARMS.....	2-15
PSMERCHBI	2-15
PSMERCHBIPARMS.....	2-16
PSMERCHANTCAT.....	2-16
PSMERCHANTAPP.....	2-16
PSSESSIONDATA	2-16
Single Sign-On PeopleCode Functions	2-17
GetAuthenticationParms	2-17
CreateSessionInformation	2-17
InsertSessionData.....	2-18
GetSessionData	2-18
DeleteSessionData.....	2-19
Establish the Merchant Account	2-19
Create the MIP	2-20
Create the Business Interlink	2-20
Create Merchant Categories.....	2-20
Create a New Merchant Profile.....	2-21
Set Up Connection and Authentication Parameters	2-22
Specify Business Interlink Information.....	2-23
Specify Alternate Merchant URLs by Category	2-24
Specify Application Attributes.....	2-24
Build or Modify the Application Page	2-24
Create the MIP PeopleCode.....	2-25

Implementing an MIP	2-30
Implementation Tasks	2-30
Implementing MIPs Delivered with Your PeopleSoft Application	2-30
Activate the Merchant Account.....	2-31
Configure the Merchant Profile.....	2-31
Modifying the Merchant Profile Page	2-32
Modifying the Merchant Authentication Page	2-32
Modifying the Merchant BI Overrides Page	2-33
Modifying the Merchant Category Page	2-33
Modifying the Application Attributes Page	2-34
Manage User Access to Merchant Services	2-34

Chapter 3

The PeopleSoft API Repository

Using the Repository	3-1
Example of Using the Repository	3-2
Repository Properties	3-7
Bindings	3-7
Namespaces.....	3-7
Bindings Collection Properties.....	3-7
Count.....	3-7
Bindings Collection Methods.....	3-8
Item	3-8
Bindings Properties	3-8
Name	3-8
Bindings Methods	3-9
Generate	3-9
Namespaces Collection Properties	3-9
Count.....	3-9
Namespaces Collections Methods.....	3-9
Item	3-9
ItemByName	3-10
Namespaces Properties.....	3-10
Classes.....	3-10
Name	3-11
Namespaces Methods.....	3-11
CreateObject.....	3-11
ClassInfo Collection Properties.....	3-11
Count.....	3-11
ClassInfo Collection Methods.....	3-12

Item	3-12
ItemByName	3-12
ClassInfo Properties	3-13
Documentation	3-13
Methods.....	3-13
Name	3-13
Properties	3-13
MethodInfo Collection Methods	3-13
Item	3-13
ItemByName	3-14
MethodInfo Collection Properties	3-14
Count.....	3-14
MethodInfo Properties	3-15
Arguments	3-15
Documentation	3-15
Name	3-15
Type	3-15
PropertyInfo Collection Methods.....	3-16
Item	3-16
ItemByName	3-16
PropertyInfo Collection Properties.....	3-17
Count.....	3-17
PropertyInfo Properties	3-17
Documentation	3-17
Name	3-17
Type	3-18
Usage.....	3-18
Example Using Visual Basic.....	3-18
Summary of Repository Methods and Properties	3-21

Chapter 4

Introducing File Layout

Fields: A Breakdown of Files	4-1
Creating a File Layout in Application Designer	4-2
Naming File Layouts, Records and Fields	4-5
Date, Time, and DateTime Field Considerations	4-6
Customizing the File Layout	4-7
File Layout Properties	4-7
File Record Properties.....	4-9
File Field Properties	4-12

Using Segments instead of Records.....	4-16
File Layout Example.....	4-16
Supported File Formats.....	4-25
Fixed Format Positional File	4-25
Considerations using FIXED Format	4-26
Variable Format Delimited File (CSV).....	4-26
Considerations using CSV Format.....	4-27
Tagged Hierarchical Data File (XML).....	4-27
Considerations using XML Format.....	4-28

Chapter 5

Open Query ODBC Driver and API

Overview.....	5-1
Features	5-1
Architecture.....	5-2
Components.....	5-3
Open Query ODBC Driver.....	5-3
Open Query API.....	5-3
External Reporting Tools	5-4
Internet	5-4
Supported ODBC v2.5 Functions	5-4
ODBC Driver Application Flow.....	5-6
Initializing PeopleTools.....	5-6
SQLAllocEnv	5-6
SQLAllocConnect	5-7
SQLFreeEnv.....	5-7
Connection Model	5-7
ODBC API Functions	5-8
SQLConnect.....	5-8
SQLDriverConnect	5-8
PeopleSoft Driver.....	5-9
Information Procedures.....	5-10
SQLGetInfo.....	5-10
SQLFunctions	5-11
SQLGetTypeInfo.....	5-11
Catalog Procedures (Meta data)	5-12
SQLProcedures	5-12
SQLProcedureColumns.....	5-13
PeopleSoft Driver.....	5-14
Executing SQL	5-14

Statement Handle	5-15
SQLAllocStmt	5-15
Execution Models	5-15
SQLExecDirect	5-16
SQLPrepare	5-17
SQLExecute	5-17
Descriptive Information	5-18
SQLColAttributes	5-18
SQLDescribeCol	5-18
SQLDescribeParam	5-19
SQLGetRowCount	5-19
SQLNumParams	5-20
SQLNumResultCols	5-20
Binding Application Data	5-20
SQLBindCol	5-21
SQLBindParameter	5-21
Literal Parameters	5-22
Retrieving Results	5-22
Retrieving One Value Directly	5-22
SQLFetch	5-23
SQLGetData	5-23
Retrieving Status and Error Information	5-24
SQLError	5-24
Terminating Transactions and Connections	5-25
SQLTransact	5-25
SQLDisconnect	5-25
SQLFreeConnect	5-25
SQLFreeEnv	5-26
ODBC Compliance	5-26
Core API	5-26
Level 1 API	5-27
Level 2 API	5-27
ODBC to RDM Data Types	5-27
Example Using the Open Query ODBC API	5-28

Chapter 6

PeopleTools Command Line Parameters

Command Line for Start	6-1
Command Line Parameters	6-2
Examples	6-4

Command Line for Project Build.....	6-4
Example	6-5
Command Line for Upgrade Copy	6-6
Example	6-9
Object Type Selection Table	6-9

Chapter 7

EDI Manager

Understanding Electronic Data Interchange	7-1
Converting EDI Codes and PeopleSoft Codes	7-3
Action Codes and Event Codes	7-3
Defining Primary and Secondary Event and Action Codes	7-4
Data Values	7-6
Identifying Database Table for Conversion	7-7
Creating a Conversion Data Profile	7-8
Defining EDI Transactions	7-10
Defining Transactions	7-10
Setting Up Trading Partners	7-15
Deleting EDI Manager Objects	7-21

Chapter 8

Mapping EDI Transactions

Processing New EDI Transactions	8-1
PeopleSoft Business Document Format	8-2
Control Records	8-3
‘999’ Record Format	8-3
‘998’ Record Format	8-4
Creating Electronic Commerce Maps.....	8-4
Creating Map Profiles.....	8-20
Using the EDI Manager for General Data Extraction.....	8-22

Chapter 9

Monitoring EDI Processing

Managing EDI Agents	9-1
Run Controls	9-2
Preparing Outbound Maps	9-2
Starting EDI Agents	9-4
Viewing the EDI Audit Trail	9-11
Reviewing and Correcting Errors.....	9-12
Packages and Transaction Groups.....	9-12

Chapter 10

Message Agent

Message Agent Overview	10-1
Managing Multiple Scroll Levels (Level Mapping).....	10-2
Limitations of Multiple Scroll Levels	10-2
Adding or Updating Multiple Rows.....	10-3
Retrieving Multiple Rows	10-3
Message Agent Field Mapping	10-4
Programming the Message Agent.....	10-4
Message Agent API Setup.....	10-5
Basic program setup	10-5
Searching for Records/Search Dialog Processing.....	10-6
Edit Table Processing.....	10-8
Message Agent Examples	10-9
Add a level 0 row	10-9
Add or update a row (non-level 0)	10-11
Read a row	10-13
Add or update multiple rows (non-level 0)	10-15
Read multiple rows	10-17
Get error information	10-19
Get message definition field information.....	10-21
Get search dialog information.....	10-23
Do search dialog processing.....	10-24
Get edit/prompt table information.....	10-26
Do edit/prompt table processing	10-26
C/C++ program	10-28
Troubleshooting the Message Agent	10-31
Message Agent Debugging	10-31
Installation.....	10-31
Administration	10-32
Declaring functions	10-32
Connecting	10-33
StartMessage	10-33
ProcessMessage.....	10-33
Tips for Message Agent	10-35
Inserting multiple Level 1 rows.	10-35
Retrieving multiple output rows.....	10-35
Using %MessageAgent in PeopleCode.....	10-35
Duplicate Keys	10-36
“Specified record already exists – update?”	10-36

Performance	10-36
Message Agent API	10-36
C Function API Specifics.....	10-36
OLE Automation Specifics	10-37
Operations by Functional Category.....	10-37
Session Level Operations.....	10-37
Processing Messages.....	10-37
Field Level Operations.....	10-38
Error Processing.....	10-39
Search Dialog Processing.....	10-39
List Box Processing.....	10-39
Edit Table Processing.....	10-40
CheckAndSetOperator	10-41
Connect	10-42
Disconnect.....	10-43
FindField	10-43
FindFirstField.....	10-44
FindFirstListBoxField	10-44
FindFirstListBoxRow	10-45
FindFirstPromptValueRow	10-46
FindListBoxField	10-46
FindListBoxRow	10-47
FindNextField	10-48
FindNextListBoxField.....	10-49
FindNextListBoxRow	10-50
FindNextOutputRow	10-51
FindNextPromptValueRow	10-51
FindPromptValueRow.....	10-52
GetEditTableFieldCount	10-53
GetEditTableFieldInfo	10-54
GetEditTableFieldList.....	10-55
GetErrorExplainText.....	10-57
GetErrorExplainTextLength.....	10-58
GetErrorFieldName.....	10-58
GetErrorFieldNameLength.....	10-59
GetErrorRecordName.....	10-60
GetErrorRecordNameLength	10-61
GetErrorText	10-61
GetErrorTextLength.....	10-62
GetFieldCount	10-63

GetFieldInfo	10-63
GetFieldList.....	10-64
GetFieldName	10-66
GetFieldNameLength	10-67
GetListBoxField	10-67
GetListBoxFieldCount	10-69
GetListBoxFieldInfo	10-69
GetListBoxRow.....	10-70
GetListBoxRowCount.....	10-71
GetMaxFieldNameLength.....	10-72
GetMaxValueLength.....	10-72
GetPromptValueFieldCount.....	10-73
GetPromptValueInfo	10-74
GetPromptValueRow	10-74
GetPromptValueRowCount	10-75
GetSearchFieldCount	10-76
GetSearchFieldInfo	10-77
GetSearchList.....	10-78
GetSearchRecord.....	10-79
GetSearchRecordLength	10-80
GetValue	10-81
GetValueLength	10-81
ProcessMessage.....	10-82
ProcessPromptTable.....	10-83
ProcessSearchDialog.....	10-84
SetField	10-85
SetOptions	10-86
StartMessage	10-87

Chapter 11

Using Database Agents and Message Definitions

Understanding Database Agents.....	11-1
Monitoring the Database	11-2
Triggering Events through the Message Agent.....	11-2
Creating a Batch of Online Processes	11-3
Adding Database Agents to Your Workflow.....	11-4
Running Database Agents.....	11-6
Adding Database Agents to the Process Scheduler	11-7
Starting Database Agents.....	11-12
Assigning Database Agents to Components	11-12

Case Study: Remote Report Delivery	11-13
The Query	11-13
The Message Definition	11-13
The Component	11-14
Troubleshooting the Database Agent	11-14
Understanding the Message Agent and Message Definitions	11-15
Message Agent	11-15
Message Definitions	11-16
Creating Message Definitions	11-16

Chapter 12

Outgoing Forms API

Understanding Forms Routings	12-1
PSFORMS.DLL	12-2
Forms API	12-3
Operations by Functional Category	12-3
Session Level Operations	12-3
Query Operations	12-3
Send Operation	12-3
PsfCloseSession	12-3
PsfGetAPIInfo	12-4
PsfGetFieldCount	12-5
PsfGetFieldList	12-5
PsfGetFormCount	12-7
PsfGetFormList	12-7
PsfGetLastError	12-8
PsfOpenSession	12-9
PsfSendForm	12-9

Index

ABOUT THIS PEOPLEBOOK

PeopleSoft provides an enterprise suite of business applications that, in some cases work with other applications in an integrated fashion. Integration Tools describes some of the ways in which you can link PeopleSoft applications and third-party applications together. This book includes the following:

Overview of Integration Technologies gives an overview of various PeopleSoft integration tools and when to use them. It also introduces the PeopleSoft Integration Software Development Kit provided with PeopleTools, including a sample project for integration development testing.

Merchant Integration gives an overview of PeopleSoft's Merchant Integration system and how to develop and implement a merchant integration point.

The PeopleSoft API Repository describes the internal classes, methods, and properties provided by PeopleSoft for integration.

Introducing File Layout describes the mapping of fields in a file and how to create a file layout in Application Designer.

Open Query ODBC Driver and API describes the Open Query Interface for PeopleTools. Beginning with an overview of the product requirements, this document will describe an API based on the defacto data access standard ODBC.

PeopleTools Command Line Parameters lists the parameters you can add to the command lines that start PeopleTools. These parameters enable you to specify login information and to automatically navigate to specific panels.

EDI Manager introduces the PeopleSoft application you use to control the exchange of electronic data interchange (EDI) transactions. It provides an overview of our EDI architecture and explains how to provide EDI setup information for your company and its trading partners.

Mapping EDI Transactions describes the process by which EDI Agents transfer EDI transaction data into the PeopleSoft database. It explains how to create maps that specify how the EDI Agent translates between PeopleSoft Business Documents, which are text files, and the PeopleSoft database tables.

Monitoring EDI Processing describes the administration processes for EDI. It tells you how to schedule EDI Agents to run and how to review audit trail information.

Message Agent covers the Message Agent process and programming the Message Agent APIs for communicating with the PeopleSoft Message Agent. You use the Message Agent to exchange data with PeopleSoft applications.

Using Database Agents and Message Definitions describes how to create and use database agents to automate many routine system tasks.

Outgoing Forms API explains how to integrate an electronic forms package so that it can accept workflow routings from PeopleSoft applications. It includes function descriptions for the application programming interface (API) calls that PeopleSoft applications make.

Before You Begin

To benefit fully from the information covered in this book, you need to have a basic understanding of how to use PeopleSoft applications. We recommend that you complete at least one PeopleSoft introductory training course.

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

Related Documentation

To add to your knowledge of PeopleSoft applications and tools, you may want to refer to the documentation of the specific PeopleSoft applications your company uses. You can access additional documentation for this release from PeopleSoft Customer Connection (www.peoplesoft.com). We post updates and other items on Customer Connection, as well. In addition, documentation for this release is available on CD-ROM and in hard copy.



Important! Before upgrading, it is *imperative* that you check PeopleSoft Customer Connection for updates to the upgrade instructions. We continually post updates as we refine the upgrade process.

Documentation on the Internet

You can order printed, bound versions of the complete PeopleSoft documentation delivered on your PeopleBooks CD-ROM. You can order additional copies of the PeopleBooks CDs through the Documentation section of the PeopleSoft Customer Connection Web site:
<http://www.peoplesoft.com/>

You'll also find updates to the documentation for this and previous releases on Customer Connection. Through the Documentation section of Customer Connection, you can download files to add to your PeopleBook library. You'll find a variety of useful and timely materials, including updates to the full PeopleSoft documentation delivered on your PeopleBooks CD.

Documentation on CD-ROM

Complete documentation for this PeopleTools release is provided in HTML format on the PeopleTools PeopleBooks CD-ROM. The documentation for the PeopleSoft applications you have purchased appears on a separate PeopleBooks CD for the product line.

Hardcopy Documentation

To order printed, bound volumes of the complete PeopleSoft documentation delivered on your PeopleBooks CD-ROM, visit the PeopleSoft Press Web site from the Documentation section of PeopleSoft Customer Connection. The PeopleSoft Press Web site is a joint venture between PeopleSoft and Consolidated Publications Incorporated (CPI), our book print vendor.

We make printed documentation for each major release available shortly after the software is first shipped. Customers and partners can order printed PeopleSoft documentation using any of the following methods:

Internet

From the main PeopleSoft Internet site, go to the Documentation section of Customer Connection. You can find order information under the Ordering PeopleBooks topic. Use a Customer Connection ID, credit card, or purchase order to place your order.

PeopleSoft Internet site: <http://www.peoplesoft.com/>.

Telephone

Contact Consolidated Publishing Incorporated (CPI) at **800 888 3559**.

Email

Email CPI at callcenter@conpub.com.

Typographical Conventions and Visual Cues

To help you locate and interpret information, we use a number of standard conventions in our online documentation.

Please take a moment to review the following typographical cues:

`monospace font`

Indicates PeopleCode.

Bold

Indicates field names and other page elements, such as buttons and group box labels, when these elements are documented below the page on which they appear. When we refer to these elements elsewhere in the documentation, we set them in Normal style (not in bold).

We also use boldface when we refer to navigational paths, menu names, or process actions (such as **Save** and **Run**).

Italics

Indicates a PeopleSoft or other book-length publication. We also use italics for *emphasis* and to indicate specific field values. When we cite a field value under the page on which it appears, we use this style: *field value*.

We also use italics when we refer to words as words or letters as letters, as in the following: Enter the number 0, not the letter O.

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 W.

Jump links

Indicates a jump (also called a link, hyperlink, or hypertext link). Click a jump to move to the jump destination or referenced section.

Cross-references

The phrase For more information indicates where you can find additional documentation on the topic at hand. We include the navigational path to the referenced topic, separated by colons (:). Capitalized titles in *italics* indicate the title of a PeopleBook; capitalized titles in normal font refer to sections and specific topics within the PeopleBook. Cross-references typically begin with a jump link. Here's an example:

For more information, see Documentation on CD-ROM in *About These PeopleBooks*: Related Documentation.

• Topic list

Contains jump links to all the topics in the section. Note that these correspond to the heading levels you'll find in the Contents window.



Name of Page or
Dialog Box

Opens a pop-up window that contains the named page or dialog box. Click the icon to display the image. Some screen shots may also appear inline (directly in the text).



Text in this bar indicates information that you should pay particular attention to as you work with your PeopleSoft system. If the note is preceded by **Important!**, the note is crucial and includes information that concerns what you need to do for the system to function properly.



Text in this bar indicates For more information cross-references to related or additional information.



Text within this bar indicates a crucial configuration consideration. Pay very close attention to these warning messages.

Comments and Suggestions

Your comments are important to us. We encourage you to tell us what you like, or what you would like changed about our documentation, PeopleBooks, and other PeopleSoft reference and training materials. Please send your suggestions to:

PeopleTools Product Documentation Manager
PeopleSoft, Inc.
4460 Hacienda Drive
Pleasanton, CA 94588

Or send comments by email to the authors of the PeopleSoft documentation at:

DOC@PEOPLESOFT.COM

While we cannot guarantee to answer every email message, we will pay careful attention to your comments and suggestions. We are always improving our product communications for you.

CHAPTER 1

Overview of Integration Technologies

The following technology offerings from PeopleSoft provide a complete toolkit to integrate your internet, third party, legacy, and custom in-house applications. This section provides a short overview of each technology – for more details, refer to the book or chapter covering that technology.

Application Messaging

Application Messaging—also referred to as Publish and Subscribe (Pub/Sub)—enables applications to publish data that can be subscribed to by multiple processes. These processes can be located on the same database and/or server or can be located on another database and/or server. It works in an asynchronous environment, which means the publisher doesn't need to be connected to the subscriber when publishing the data. This is comparable to the way email uses queues in order to guarantee the delivery of a message to its subscribers. Wherever possible, interfaces across databases have been designed to use an asynchronous interface via Application Messaging. This provides customers with more flexibility in deploying and operating PeopleSoft systems.

When to Use Application Messaging

- **Data synchronization.** Application Messaging can help to synchronize data stored in different systems. For example, imagine that an employee changes job codes, and that this change is entered into a PeopleSoft HRMS application. As soon as this change is entered and saved, the HRMS system publishes a message containing the changed data on the messaging network. Other systems on the network, such as manufacturing or accounting systems, may have subscribed to messages of this type.
- **System-to-system workflow.** You can use application messaging as an extension of PeopleSoft workflow functionality. PeopleSoft workflow is based on business events: conditions that trigger the need for a follow-up activity. Business events can direct work from person to person and also from system to person. Application messaging enables system-to-system workflow by eliminating the need to programmatically drive the execution of each step of a business process spanning different systems. Instead of creating a custom program to drive workflow, the delivery of an application message to a subscriber initiates the execution of the next step through the use of a *subscription process*. For example, hiring an employee in PeopleSoft HRMS could create a purchase requisition for supplies or a worklist entry for a supplies administrator.
- **Asynchronous integration.** When you need to integrate with our third party partners in an

asynchronous environment. You do not want to use application messaging in scenarios where the publishing system needs to wait for an "answer" from another system; in this situation, use business interlinks.



For more information about application messaging, refer to PeopleSoft Application Messaging.

Component Interfaces

Component Interfaces (CI) expose the rich functionality delivered in the hundreds of components that make up PeopleSoft products. A component is an atomic transaction which implements a business process or function. A component interface provides real time synchronous access to the PeopleSoft business rules and data associated with a business component. The interface is exposed via standard access methods. Component Interfaces can be viewed as "black boxes" that encapsulate PeopleSoft data and business processes, and hide the details of the structure and implementation of the underlying page and data. The actual interface consists of a set of clearly defined properties and methods that follow an object-oriented programming model. External applications can only access a component's data using the interface's specified properties or methods.

When to Use Component Interfaces

- When third parties must retrieve and/or update PeopleSoft data real time using request/reply synchronously
- When you want to recycle the online business logic associated with a PeopleSoft component



For more information about component interfaces, refer to PeopleSoft Component Interface.

Business Interlink

PeopleSoft Business Interlink is an integrated framework that allows PeopleSoft applications to access, update and invoke procedures in an external system in a real-time synchronous mode using a generic in-memory API. Prior to PeopleTools 8, calling out to an external system from PeopleCode required either writing a vendor-specific DLL on Windows NT or writing shared libraries on Unix, and the corresponding PeopleCode to invoke the routine. Business interlinks enable a PeopleCode program to map the inputs and outputs of an external system to PeopleCode variables and to PeopleSoft record fields.

Each business interlink definition represents a business transaction associated with an integration point; for example, calculate sales tax. Each interlink definition is associated with an *interlink plug-in*, created in XML and C++ by PeopleSoft or by a third party associated with the external

system. Interlink plug-ins are self-describing class libraries that expose their services to the Application Designer and contain the runtime routines for executing the interface to the external system. Since the connection between systems occurs through the business interlink, neither system needs to know the internal workings of the other.

The business interlink framework provides for querying data, updating data, adding data, deleting data, and performing specific transactions against objects in an external system. It can process a single transaction in a real-time fashion, or it can process a batch of data by loading multiple rows of input into its input buffer and then invoking the corresponding method.

When to Use Business Interlinks

- When you need to access or update data in an external system from PeopleCode in a real time synchronous environment.
- When you need to invoke a function or procedure in an external system from PeopleCode in a real time synchronous environment.
- When you need to call C++ routines to perform specific functions or calculations that can not be effectively written in PeopleCode.



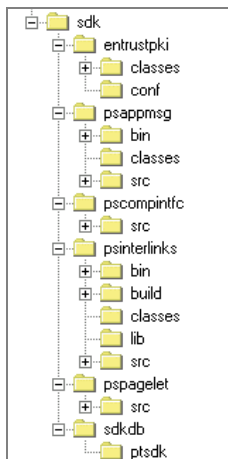
For more information about business interlinks, refer to PeopleSoft Business Interlink Application Developer Guide.

Integration Software Development Kit

The PeopleSoft Integration Software Development Kit (SDK) consists of utilities, sample data and source code for use with PeopleSoft's application messaging, component interface, business interlink and portal pagelet technologies, along with a sample database project you can use to test your work. These technologies are recommended as the basis for your integrations, but PeopleSoft 8 includes several other tools and technologies you can use, described elsewhere in this PeopleBook.

PeopleSoft applications are delivered with many completely developed integrations implemented for common activities in a variety of categories. You can use them as templates for your own integration development. The SDK assumes knowledge of one or more of the following languages: PeopleCode, C++, Visual Basic (VB), Active Server Pages (ASP) or XML, depending on the technology used.

The SDK is automatically installed to the PeopleSoft home directory (PS_HOME), under **sdk**.



Integration SDK directory structure

Related Documentation

Use the following documentation as your primary resource for developing, testing and implementing integrations:

- The *Enterprise Integration PeopleBook* is your most comprehensive resource for developing integrations between applications, called Enterprise Integration Points (EIPs). It includes information about development tools and techniques, and the EIPs delivered with PeopleSoft applications.
- If you want to incorporate third party merchant services into your application over the internet, PeopleSoft provides a framework for quickly developing and managing Merchant Integration Points (MIPs). Refer to the overview of Merchant Integration for more information.
- The PeopleSoft API Repository allows PeopleCode and third party integrators to discover the internally available classes, methods, and properties provided by PeopleSoft for integration. The API Repository documentation explains how to use this facility.
- File layouts enable you to read and write complex structured data between your PeopleSoft application and third party products using ordinary text files. Refer to the overview of File Layouts for more information.
- PeopleSoft provides an implementation of the Entrust public key infrastructure to enable SSL connections between HTTP servers. The Entrust PKI module is certified Entrust-Ready™, and is provided as part of the Integration SDK. You can use it to replace the default keystore shipped as part of your PeopleSoft application. You can find installation instructions for the Entrust PKI module in `<PS_HOME>\sdk\entrustpki\readme.txt`.

The PeopleBook documentation for the following technologies includes sections covering the use of the Integration SDK in developing and testing integrations with each technology:

- Application messaging — refer to the overview of Application Messaging for more information.
- Component interfaces — refer to the overview of Component Interfaces for more information.

- Business interlinks — refer to the overview of Business Interlinks for more information.
- Portal pagelets — refer to Developing Pagelets for more information.

File Layout Objects / Definitions

The File class provides methods and properties for reading from and writing to external files. Most application interfaces to files require complex parsing of file data. Files that allow for this kind of complexity in a PeopleSoft application are based on a file layout. A file layout is a definition (or mapping) of a file to be processed. It identifies where in the file data fields are located. This powerful interface allows application developers to access data from a file as they would a message or a panel buffer (scroll). There is no need to parse each file record into fields.

The following formats are supported by file layout definitions:

- Fixed Format Positional File (FIXED)
- Variable Format Delimited File (CSV)
- Tagged Hierarchical Data File (XML)

File layouts will be unique to a specific format, and may only process that particular type of formatted file. The definition created in the Application Designer retains a consistent look and feel irrespective of format. File layouts rely solely on PeopleCode as the engine behind the actual data access and movement.

When To Use File Layouts

- When you need to load a flat file to a table, or when you need to unload a table to a flat file



For more information about file layouts, refer to [Introducing File Layout](#).

EDI Manager

The EDI (Electronic Data Interchange) Manager is a tool for managing a customer's electronic commerce transactions with their trading partners. It provides panels that allow customers to setup and maintain data about their trading partners, define additional e-commerce transactions based on their trading partner's requirements and monitor EDI processing activities. The batch inbound and outbound transaction processing of EDI Manager are written in Application Engine and use a grid to map data between the tables in a PeopleSoft application and PeopleSoft Business Documents. PeopleSoft Business Documents are then typically converted by a third party to the standard X.12 or EDIFACT EDI format.

When To Use EDI Manager

- When you need to define a PeopleSoft Business Document which will be used for an X.12 or EDIFACT EDI transaction.



Note. EDI Manager should no longer be used to load or unload flat files that are not X.12 or EDIFACT EDI transactions. File layouts have stronger file access methods.



For more information about EDI Manager, refer to EDI Manager.

Merchant Integration

PeopleSoft's Merchant Integration capability enables end users to interact with third-party Web sites and other Internet-based services directly from a PeopleSoft application, effectively making those merchant services part of the application interface. This dramatically enhances the application's functionality and the end user's experience.

Using a subset of the PeopleTools integration technologies, you can develop a **Merchant Integration Point (MIP)** to incorporate a merchant's services into your application. An MIP communicates with the merchant's Website to enable the merchant to act in a PeopleSoft application support role, facilitating a variety of interactions between your end users and the merchant.

An MIP adds a merchant's services to a PeopleSoft application page, either as an underlying transactional exchange triggered by the user's interaction with the application, or in the form of an interface branded with the merchant's logo that enables a user to interact directly or indirectly with the merchant. In the delivered software, the user is typically presented with a choice of merchant services on a PeopleSoft community page, such as purchasing supplies, making travel arrangements, or accessing health plan information.

The user interacts with the PeopleSoft interface, which communicates with the merchant's server, producing an integrated environment. Many of the elements used to accomplish this exchange are part of the PeopleSoft Internet Architecture (PIA) and the Single Sign-On Framework delivered with your application.

PeopleSoft's Single Sign-On Framework addresses the need for a third-party merchant to seamlessly authenticate a user who initiates business transactions using an MIP from a PeopleSoft application page. The Single Sign-On Framework comprises all the elements needed for developing MIPs with this functionality, including the Merchant Categories and Merchant Profile components, the generic business interlink run-time plug-in for HTTP-based transactions (HTTPEnable.dll), and all the records and pre-defined PeopleCode functions required for assembling these elements into a typical MIP.

When To Use Merchant Integration

- To integrate third party services which are accessible over the Internet into your PeopleSoft application.



For more information about merchant integration and the Single Sign-On Framework, see Merchant Integration.

Message Agent

The Message Agent processes real time synchronous messages sent to our applications by an external system, such as a Visual Basic or C program, and uses the existing business rules associated with a PeopleSoft component. It can send data to and receive data from the component.

When To Use Message Agent



Note. Component interfaces are a more robust alternative for exposing business rules to external applications that require a real-time synchronous connection. The Message Agent is used to support certain functionality from previous releases, but should not be used for new development.



For more information about Message Agent, refer to Message Agent.

Database Agents

A database agent is a workflow program that monitors one or more tables in the database for conditions that should trigger business events. Database agents use a database agent query, which is a special type of PeopleSoft Query, to determine the data that needs to be processed and pass the results of the query to the Message Agent. If the component that the Message Agent connects to has a business event associated with it, that event is triggered when the data is saved. The database agent is scheduled to run on a periodic basis using the Process Scheduler.

When To Use Database Agents



Note. Database agents are used in conjunction with the Message Agent to support certain functionality from previous releases, but should not be used for new development. You should choose a more advanced technology based on your business requirements. One option, for example, is to write an Application Engine program that queries the database and calls a component interface.



For more information about database agents, refer to *Using Database Agents and Message Definitions*.

Merchant Integration

Understanding Merchant Integration

In PeopleSoft 8, PeopleSoft's Merchant Integration capability enables end users to interact with third-party Web sites and other Internet-based services directly from a PeopleSoft application, effectively making those merchant services part of the PeopleSoft interface. This dramatically enhances the end user's experience of the application.

Merchant integration is accomplished using **Merchant Integration Points (MIPs)**. An MIP is an enhanced form of Enterprise Integration Point (EIP), but instead of connecting to another application, it incorporates a third-party merchant's services into your PeopleSoft application using a combination of page controls, PeopleTools integration technologies, and PeopleCode. It enables a merchant whose services you purchase to act in a PeopleSoft application support role, and it can facilitate a variety of interactions between your end users and the merchant.

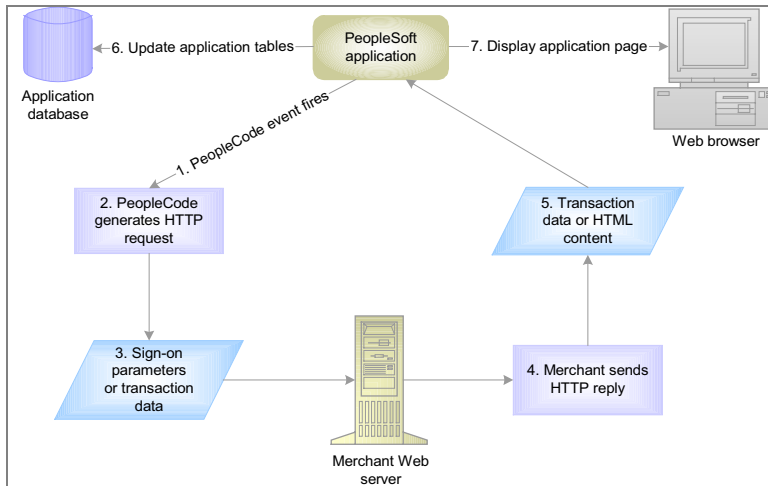


Although they use some of the same technologies, EIPs are developed and used differently from MIPs, and are documented separately. For more information about EIPs, see the Enterprise Integration PeopleBook.

If you're implementing one or more MIPs that were delivered with your PeopleSoft application, skip the section in this chapter titled "Developing an MIP," where we discuss the technical details of Merchant Integration. To learn more about the MIPs delivered with your application, see your application documentation. If you want to develop your own MIP, read this entire chapter.

Merchant Integration Process Flow

An MIP adds a merchant's services to a PeopleSoft application page, either as an underlying transactional exchange triggered by the user's interaction with the application, or in the form of an interface that enables a user to interact directly or indirectly with the merchant. The user interacts with the PeopleSoft interface, which communicates with the merchant's server, producing a seamless environment. The following diagram illustrates the basic MIP process flow.



Merchant integration process flow

In response to a PeopleCode event (1), your PeopleSoft application's PeopleCode sends an HTTP request (2) containing either sign-on parameters or transaction data (3) to the merchant's Web server. The merchant's server processes the request, then sends an HTTP reply (4) containing either transaction data or HTML content (5) back to the calling PeopleCode, which updates the application tables (6), modifies the application page content, and refreshes the page (7) as needed.

Many of the elements used to accomplish this exchange are part of the PeopleSoft Internet Architecture (PIA) or the Single Sign-On Framework delivered with your application.



For more information, see MIP Features and PeopleSoft Internet Architecture Administration.

The User Experience

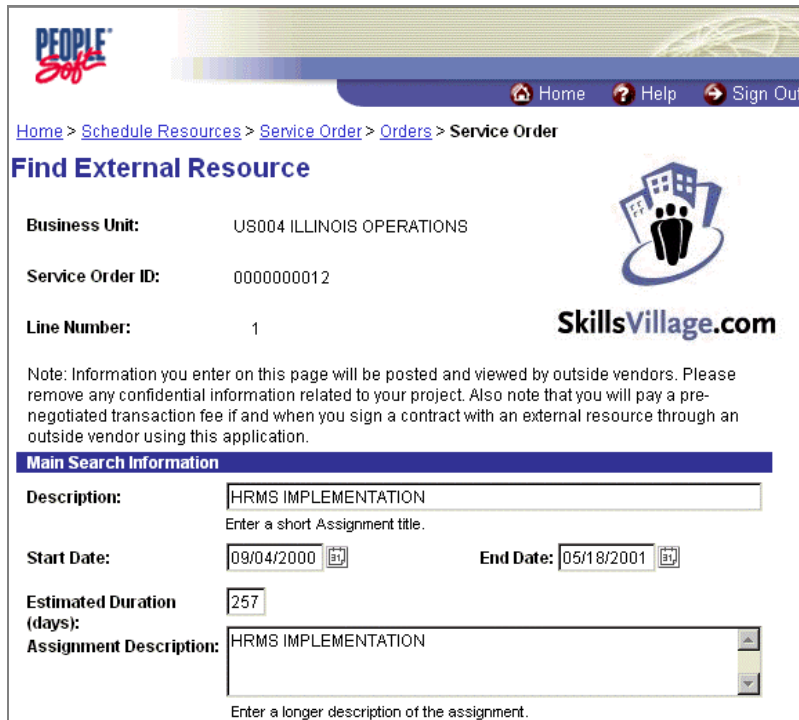
With Merchant Integration, the user is typically presented with a choice of merchant services on a PeopleSoft community page, such as purchasing supplies, making travel arrangements, or accessing health plan information. Most delivered MIPs launch from a PeopleSoft community page, but you can develop new MIPs that launch from any application page. An MIP can be made available to users in several ways, as described by the following configurations.

Hidden Application Functionality

The merchant provides functionality that underlies an application, transparent to the user. The MIP's functionality is integrated into the application page using PeopleCode. A user action, such as saving the page, triggers a transaction between the application and the merchant site. Information about the transaction and its results may or may not be provided to the user.

Branded Application Functionality

In this scenario, the merchant is presented as a branded portion of a PeopleSoft application page. Part or all of the application page consists of a PeopleSoft interface dedicated to interacting with the merchant site. The merchant's brand name and logo appear on the page, but the page controls and layout are part of the PeopleSoft application. The controls trigger MIPs that submit transactions to the merchant site, retrieve the merchant's responses, and insert the responses into the page's data display. This is the most common type of MIP interface.



PEOPLE
Soft

Home Help Sign Out

Home > Schedule Resources > Service Order > Orders > Service Order

Find External Resource

Business Unit: US004 ILLINOIS OPERATIONS

Service Order ID: 0000000012

Line Number: 1

SkillsVillage.com

Note: Information you enter on this page will be posted and viewed by outside vendors. Please remove any confidential information related to your project. Also note that you will pay a pre-negotiated transaction fee if and when you sign a contract with an external resource through an outside vendor using this application.

Main Search Information

Description: HRMS IMPLEMENTATION
Enter a short Assignment title.

Start Date: 09/04/2000 **End Date:** 05/18/2001

Estimated Duration (days): 257

Assignment Description: HRMS IMPLEMENTATION
Enter a longer description of the assignment.

Top of application page with branded merchant service

Merchant HTML Content

Here the merchant responds to the MIP by providing HTML content as the basis for all or part of the application page content. Your PeopleSoft application may reformat the content to generate a presentation consistent with the rest of the application, but the controls on the page are provided by the merchant. By interacting with the page, the user interacts directly with the merchant site. The application can use or store the information on the page just as with any other PeopleSoft page.

MIP Features

Installing MIPs

PeopleSoft applications are delivered with a number of MIPs already developed and available for use. When you install an application, the components for all MIPs developed for that application

are installed as well, with their functionality disabled by default. After installing your application, you can decide which MIPs to enable, then determine which users will have access to the MIPs you've enabled. You must establish an appropriate account with each merchant whose MIP you enable, so the MIP can access the merchant's Web site.



For more information about enabling an MIP, establishing a merchant account, and managing user access to merchant services, see *Implementing an MIP*.

Support

PeopleSoft supports the portion of the MIP that's part of your PeopleSoft application. Merchants are responsible for the proper operation of their Web sites, and for responding properly to requests directed to them by your PeopleSoft application.

Single Sign-On Framework

PeopleSoft's Single Sign-On Framework addresses the need for a third-party merchant to seamlessly authenticate a user who initiates business transactions using an MIP from a PeopleSoft application page. With Single Sign-On, the user has to sign on only once, when entering the PeopleSoft application. The need for a subsequent sign-on to the merchant system is handled without directly involving the user, who doesn't have to remember multiple user IDs or passwords. This enables the user to make transactions on a merchant system without having the sense of moving from one application to another. To the user, all transactions occur within the PeopleSoft application.

The Single Sign-On Framework comprises all the elements needed for developing MIPs with this functionality, including the Merchant Categories and Merchant Profile components, the generic business interlink run-time plug-in for HTTP-based transactions (HTTPEnable.dll), and all the records and pre-defined PeopleCode functions required for assembling these elements into a typical MIP. The Single-Sign-On Framework makes it easy to modify and implement an MIP.

For a given MIP, you establish a single company account for each desired service for which the merchant requires a distinct sign-on ID and password. When a user triggers the MIP, your PeopleSoft application signs on to the merchant site. The sign-on ID it uses can be any agreed upon identification, but one benefit of the Single Sign-On Framework is that you can use a pre-defined Company ID and password which signs on to a single company account regardless of the current user. The MIP can also pass the merchant a user-specific ID for uniquely identifying and maintaining account information for the current user during the current session. This Merchant User ID is automatically generated by the MIP based on criteria you specify in the merchant profile.



For more information about how to create an MIP using Single Sign-On, see *Developing an MIP*. For more information about the merchant profile, see *Maintaining Merchant Information*. For more information about managing access to a merchant, see *Manage User Access to Merchant Services*.

Secure Sockets Layer

Secure Sockets Layer (SSL) is the most widely used security protocol on the Web. An MIP should use SSL for key activities such as sign-on and credit card transactions, but doesn't require it otherwise. The run-time plug-in HTTPEnable.dll handles SSL automatically.



For more information about creating business interlinks using HTTPEnable.dll, see PeopleSoft Business Interlink Application Developer Guide and Writing a XML Design-Time Plug-In using the pshttpenable Runtime Plug-In.

Session Information

The Single Sign-On Framework provides predefined PeopleCode functions to support you in monitoring and storing data for multiple merchant sessions. It provides an application data table to contain the session IDs and machine names for multiple merchants at PeopleCode run time. You can use the functions to record, retrieve and delete session information in that table. The table is global, so it will be accessible from PeopleCode at all times while the user is in the application, and it tracks all merchant systems to which the user is signed on. Once the user exits the application, the session information is discarded.



For more information about merchant sessions, see Single Sign-On PeopleCode Functions and Create the MIP PeopleCode.

Maintaining Merchant Information

Merchant information is stored in the Merchant Categories and Merchant Profile components; you'll use these components both for developing a new MIP and for implementing a delivered MIP.



For more information about using merchant categories and profiles, see Create the MIP and Implementing an MIP.

Merchant Categories Component

This component consists of one page, the Merchant Categories page. It serves as a repository of predefined categories that can be used in a merchant profile. Each category represents a type of service that a merchant might provide. A merchant can be assigned to multiple categories, or to no category.

Merchant Categories Page

Usage	Use the Merchant Categories page to list all available merchant categories.
Object Name	MERCHANTCATTBL
Navigation	Go, PeopleTools, Utilities, Use, Merchant Categories
Prerequisites	None
Access Requirements	Enter a Merchant Category ID.
Where Used	Merchant Category page.

Merchant Categories

Merchant Category: RECRUIT

Effective Date	Status	Description	Short Description
08/01/2000	Active	Recruit Contractors	Recruiting

Buttons: Save, Add, Update/Display, Include History, Correct History

Merchant Categories page

Effective Date

Enter the date on which you want the displayed row to become the current row.

Status

Select one of the following values:

Active: Select this to enable the displayed row to become the current row on its effective date.

Inactive: Select this to disable the displayed row. An inactive row will never become the current row, regardless of its effective date.

Description

Enter a description of the category that's general enough to encompass similar services from different merchants you may assign to the category.

Short Description

Enter a short description of the category.



For more information about using the Merchant Categories page, see [Create Merchant Categories](#).

Merchant Profile Component

The Merchant Profile component consists of several pages that maintain the information your PeopleSoft application needs for implementing an MIP using Single Sign-On, including:

- MIP enabled status.
- Merchant contact information for setting up and managing your account.
- Your company ID and encrypted password for merchant sign-on.
- User identification for session management.
- Merchant Web site URLs for sign-on and other transactions.
- Information for overriding business interlink parameter defaults.
- Categories of services the merchant provides.
- Application specific features of the MIP.

You'll need to create a new merchant profile in order to develop a new MIP, and you'll need to modify an existing merchant profile to reflect your account information before you implement a delivered MIP. Your PeopleSoft application requires the profile information so that it can interact correctly with the merchant.

Merchant Profile Page

Usage	Use the Merchant Profile page to maintain basic information about each merchant.
Object Name	MERCHANTID
Navigation	Go, PeopleTools, Utilities, Use, Merchant Profile, Merchant Profile
Prerequisites	None
Access Requirements	Enter a merchant ID for which you want to manage a merchant integration.

Merchant ID: SKILVILG

Merchant Information View All First 1 of 1 Last

Effective Date: 08/01/2000 ☒ Merchant Enabled

Description: Skills Village

Short Description: Skills

Object owner identifier: PeopleTools

Merchant Logo: SKILLSVILLAGE_LOGO

Relationship Manager: Foreman, Fred

Relationship Manager Email: fredf@skillsvillage.com

Relationship Manager Phone: 925-555-1212

Save Previous tab Next tab Add Update/Display Include History Correct History

Merchant Profile | Merchant Authentication | Merchant BI Overrides | Merchant Category | Application Attributes

Merchant Profile page

Effective Date	Enter the date on which you want the displayed row to become the current row.
Merchant Enabled	Select this check box to enable MIPs using this profile to connect to the merchant.
Description	Enter a description of this profile that's general enough to encompass all the services you'll access using it.
Short Description	Enter a short description of the profile.
Object owner identifier	Select from a drop down list the PeopleSoft product with which MIPs using this profile are associated.
Merchant Logo	Enter the Image Catalog name of the merchant logo image you want to display in HTML areas of MIPs using this profile.
Relationship Manager	Enter the name of the merchant's support representative who serves as your company's primary contact, in standard PeopleSoft name format.
Relationship Manager Email	Enter the relationship manager's email address, in standard PeopleSoft email address format.
Relationship Manager Phone	Enter the relationship manager's telephone number, in standard PeopleSoft telephone format for your country.



For more information about using the Merchant Profile page, see [Create a New Merchant Profile and Modifying the Merchant Profile Page](#).

Merchant Authentication Page

Usage	Use the Merchant Authentication page to maintain information that your application uses to sign on to the merchant's Web site.
Object Name	MERCHANTAUTH
Navigation	Go, PeopleTools, Utilities, Use, Merchant Profile, Merchant Authentication
Prerequisites	None
Access Requirements	Enter a merchant ID for which you want to manage a merchant integration.

Merchant Profile | Merchant Authentication | Merchant BI Overrides | Merchant Category ▶

Merchant ID: SKILVILG

Merchant Information View All First 1 of 1 Last

Effective Date: 08/01/2000

*Merchant User ID Type: Operator ID + Company ID

*Company ID: MYCO

Company Authentication Token: ***** Confirm: *****

Merchant Website URL: http://www.skillsvillage.com

Save Previous tab Next tab Add Update/Display Include History Correct History

[Merchant Profile](#) | [Merchant Authentication](#) | [Merchant BI Overrides](#) | [Merchant Category](#) | [Application Attributes](#)

Merchant Authentication page

Merchant User ID Type	<p>Select one of the following values, which the merchant will use to identify a user from your company:</p> <p><i>Company Identifier:</i> Select this to use your Company ID if the merchant doesn't need to differentiate between users.</p> <p><i>Employee ID + Company ID:</i> Select this to append your Company ID to your user's employee ID.</p> <p><i>EMail Address:</i> Select this to use your user's email address. This requires that an email address has been entered into the user's PeopleSoft profile.</p> <p><i>Operator ID + Company ID:</i> Select this to append your Company ID to your user's operator ID.</p>
Company ID	Enter the name that will be used to identify your company when the MIP signs on to the merchant Web site.
Company Authentication Token	Enter the password that will be used to authenticate your company when the MIP signs on to the merchant Web site.
Confirm	This field is hidden; it appears after you enter a Company Authentication Token and press TAB. Enter the token again to Confirm it.
Merchant Website URL	Enter the complete HTTP URL you expect your MIPs to use as the default initial point of contact with the merchant Web site.



For more information about using the Merchant Authentication page, see Set Up Connection and Authentication Parameters and Modifying the Merchant Authentication Page.

Merchant BI Overrides Page

Usage	Use the Merchant BI Overrides (merchant business interlink overrides) page to specify which business interlinks to use for the current MIP, and to enter settings which will temporarily override selected default settings of any specified business interlink. When you add a row, you're prompted for a selection from the business interlink table PSIODEFN_VW.
Object Name	MERCHANTBIPARMS
Navigation	Go, PeopleTools, Utilities, Use, Merchant Profile, Merchant BI Overrides

Prerequisites	None
Access Requirements	Enter a merchant ID.

Merchant ID: SKILVILG

Merchant Information [View All](#) First 1 of 1 Last

Description: Skills Village Effective Date: 08/01/2000

Business Interlink [View All](#) First 1 of 1 Last

'Business Interlink: ☒ Sign-On Business Interlink

Business Interlink Settings [View All](#) First 1-3 of 4 Last

Override	Parameter Name	Parameter Value
<input checked="" type="checkbox"/>	Server	billyp.peoplesoft.com
<input checked="" type="checkbox"/>	Port	389
<input checked="" type="checkbox"/>	User_DN	cn=Admin,o=PeopleSoft

[Merchant Profile](#) | [Merchant Authentication](#) | [Merchant BI Overrides](#) | [Merchant Category](#) | [Application Attributes](#)

Merchant BI Overrides (merchant business interlink overrides) page

Business Interlink

Select a business interlink definition from the list of available definitions. The business interlink's parameter names and default values will appear in the **Business Interlink Settings** grid.

Sign-On Business Interlink

Select this check box to designate this business interlink as the one used to sign on to the merchant Web site. You can select this check box for only one row, or for none.

Business Interlink Settings

Select the **Override** check box to indicate that the parameter's assigned value will override its default value. When you save the page, only rows with this check box selected will remain.

Enter a new **Parameter Value** that will override the parameter's default value at run-time. This doesn't affect the stored default value for the parameter.

Important! Some business interlink setting parameters require a specific value, and should not be overridden. You're responsible for understanding which parameters you can safely override.

Select All	Click this button to select the Override check box for all displayed rows on the grid.
Select None	Click this button to clear the Override check box for all displayed rows on the grid.
Reset	Click this button to reload all of the business interlink's parameters, and reset them to their default values on this page.

Note. Although clicking **Reset** deletes any unsaved override values, you can retain previously saved values if you exit the Merchant Profile component without saving.



For more information about using the Merchant BI Overrides page, see Specify Business Interlink Information and Modifying the Merchant BI Overrides Page.

Merchant Category Page

Usage	Use the Merchant Category page to build a list of the categories to which the current merchant belongs. When you add a row, you're prompted for a selection from the MERCHANTCATTBL table.
Object Name	MERCHANTCAT
Navigation	Go, PeopleTools, Utilities, Use, Merchant Profile, Merchant Category
Prerequisites	The Merchant Categories page must contain at least one entry.
Access Requirements	Enter a merchant ID.

Merchant Profile | Merchant Authentication | Merchant BI Overrides | Merchant Category

Merchant ID: SKILVILG

Merchant Information View All First 1 of 1 Last

Description: Skills Village Effective Date: 08/01/2000

Merchant Category View All First 1 of 1 Last

Merchant Category: RECRUIT

Merchant Website URL: http://www.skillrecruit.com

Description: Recruit Contractors

Short Description: Recruiting

Save Previous tab Next tab Add Update/Display Include History Correct History

Merchant Profile | Merchant Authentication | Merchant BI Overrides | Merchant Category | Application Attributes

Merchant Category page

Merchant Category

Select a merchant category from the list of available categories. The system displays the **Description** and **Short Description** you entered on the the Merchant Categories page.

Merchant Website URL

Enter the complete HTTP address of the merchant's URL that should be used for transactions in this category.



For more information about using the Merchant Category page, see Specify Alternate Merchant URLs by Category and Modifying the Merchant Category Page.

Application Attributes Page

Usage	The Application Attributes page provides a convenient location to store information relating to your MIP. It has no specific purpose—the attributes are maintained as name/value pairs, and can represent anything.
Object Name	MERCHANTAPPATTRS
Navigation	Go, PeopleTools, Utilities, Use, Merchant Profile, Application Attributes
Prerequisites	None
Access Requirements	Enter a merchant ID.

Merchant ID: SKILVILG

Merchant Information [View All](#) First 1 of 1 Last

Description: Skills Village Effective Date: 08/01/2000

Application Attributes First 1-3 of 3 Last

Attribute Name	Attribute Value		
SYSADMIN	SKILLRECRUIT01	+	-
APPDEV	SKILLRECRUIT02	+	-
QATEST	SKILLRECRUIT04	+	-

Save Previous tab Next tab Add Update/Display Include History Correct History

[Merchant Profile](#) | [Merchant Authentication](#) | [Merchant BI Overrides](#) | [Merchant Category](#) | [Application Attributes](#)

Application Attributes page

Application Attributes

Enter an **Attribute Name** identifier to represent an attribute for which an MIP using this merchant profile expects a value.

Enter as an identifier the **Attribute Value** that an MIP using this merchant profile should use for this attribute.

Click the Plus button to add a new attribute grid row after the current one.

Click the Minus button to remove the current attribute grid row.

Developing an MIP

This section explains how to develop a new Merchant Integration Point to enhance your installed PeopleSoft application. It assumes you're familiar with the core tools and technologies involved, including business interlinks, Application Designer and PeopleCode.

Developing an MIP for your PeopleSoft application consists of two distinct activities:

1. Contact the merchant whose services you wish to use, and establish an account for your organization.
2. Use PeopleTools to create the MIP that will access and interact with the merchant account you've set up.

HTTPEnable.dll

The business interlink run-time plug-in called **HTTPEnable.dll** is particularly well suited for use in transactions over the Internet, because it uses HTTP GET and POST methods to send and

receive XML-formatted data. Your MIP will be easier to develop and maintain if your merchant can conduct all transactions—including sign-on—in XML, so that you can base your entire MIP on HTTPEnable.dll.



For more information about creating business interlinks using HTTPEnable.dll, see PeopleSoft Business Interlink Application Developer Guide and Writing a XML Design-Time Plug-In using the pshttpenable Runtime Plug-In.

Single Sign-On Records and Fields

Following is a listing of some the PeopleSoft records which contain relevant data or can be used as templates in your Single Sign-On MIP PeopleCode. For more detail, examine the record definitions in Application Designer.



For more information about using these records and fields, see Create the MIP PeopleCode.

PSAUTHPARMS

This record definition is provided for use as a template from which you can instantiate a PeopleCode object to maintain authentication information extracted from the merchant profile for sign-on, including the following fields:

- **MERCHANTID:** The merchant ID, which is the primary key for the merchant profile.
- **MRCHUSERID:** The working ID used by the merchant to identify a user from your company.
- **COMPANYID:** The ID used by your company to identify your company on the merchant system.
- **WRKTOKEN:** The decrypted password used by your company to sign on to the merchant Web site.
- **SIGNONBIURL:** The URL specified for the run-time plug-in designated as the sign-on business interlink on the Merchant BI Overrides page, if one is selected.
- **MERCHWSITEURL:** The primary Web site URL specified in the merchant profile or one of its categories for initiating MIP transactions.

PSMERCHBI

This is the PeopleTools table used to select the current effective dated row from PSMERCHBIPARMS, identified by the following fields:

- **MERCHANTID:** The merchant ID, which is the primary key for the merchant profile.
- **EFFDT:** The effective date of the merchant profile's current row.

PSMERCHBIPARMS

This is the PeopleTools table for the business interlink information on the Merchant BI Overrides page, containing the following fields:

- **EFFDT:** The effective date of the merchant profile's current row.
- **IONAME:** The name of the selected business interlink.
- **IOSETTINGNAME:** The name of a setting parameter for the selected business interlink.
- **IOVALUE:** The override value for the selected setting parameter of the selected business interlink.

PSMERCHANTCAT

This is the PeopleTools table for the category information on the Merchant Category page, containing the following fields:

- **MERCHANTID:** The merchant ID, which is the primary key for the merchant profile.
- **EFFDT:** The effective date of the merchant profile's current row.
- **MERCHANTCAT:** The name of the selected merchant category.
- **MERCHWSITEURL:** The Web site URL specified for the selected merchant category.

PSMERCHANTAPP

This is the PeopleTools table for the attribute information on the Application Attributes page, containing the following fields:

- **MRCHAPPATTRNAME:** The name of an application attribute.
- **MRCHAPPATTRVALUE:** The value of the selected application attribute.

PSSESSIONDATA

This record definition is provided for use as a template from which you can instantiate a global PeopleCode rowset object. You use the Single Sign-On Framework functions to instantiate this rowset and maintain the following fields:

- **MERCHANTID:** The merchant ID, which is the primary key for this record.
- **MRCHSESSION:** The session ID obtained from the selected merchant.

- **MRCHMACHINENAME**: The machine name obtained from the selected merchant.

Single Sign-On PeopleCode Functions

The Single Sign-On Framework includes several predefined PeopleCode functions, which reside in the FieldFormula event of the PSFUNCLIB_PTIC.MRCHSIGNONFNCTN field. You'll use these functions in your MIP PeopleCode.

GetAuthenticationParms

Syntax

```
GetAuthenticationParms(merchant_id)
```

Description

GetAuthenticationParms extracts from the merchant profile all the basic parameters necessary c

to sign on to the merchant Web site.

Parameters

<i>merchant_id</i>	Specify as a string the primary key of the search record for the desired merchant profile— PSMERCHANTID.MERCHANTID.
--------------------	--

Returns

A record containing authentication parameters retrieved from the merchant profile. This record should first be instantiated from the PSAUTHPARMS record definition. The return value is *Null* if the *merchant_id* profile is not enabled for the current effective-dated row, or if the current user's profile is unavailable.

CreateSessionInformation

Syntax

```
CreateSessionInformation()
```

Description

CreateSessionInformation creates a global rowset object called &SessionInformation, based on the PSSESSIONDATA record definition. Be sure to globally declare &SessionInformation at the top of PeopleCode programs where you want to use it.

Parameters

None.

InsertSessionData

Syntax

```
InsertSessionData(session_data_record)
```

Description

InsertSessionData inserts a new row into the global rowset object &SessionInformation and copies the contents of a session data record into it.

Parameters

<i>session_data_record</i>	Specify the record identifier of the session data record you want to copy into the &SessionInformation rowset. You should first have instantiated this record from the PSSESSIONDATA record definition and populated it.
----------------------------	--

Returns

A boolean value: *True* if the record data was successfully inserted, *False* if it wasn't.

GetSessionData

Syntax

```
GetSessionData(merchant_id)
```

Description

GetSessionData retrieves a row from the global rowset object &SessionInformation and copies its contents into a session data record.

Parameters

<i>merchant_id</i>	Specify as a string the MERCHANTID field of the row you wish to retrieve from the &SessionInformation rowset.
--------------------	---

Returns

A record containing the session data retrieved from &SessionInformation for the specified merchant. This record should first be instantiated from the PSSESSIONDATA record definition. The return value is *Null* if a record containing the MERCHANTID field doesn't exist in the rowset. Updating the return value effectively updates that row in &SessionInformation.

DeleteSessionData

Syntax

```
DeleteSessionData (merchant_id)
```

Description

DeleteSessionData deletes the specified row from the global rowset object &SessionInformation.

Parameters

<i>merchant_id</i>	Specify as a string the MERCHANTID field of the row you wish to delete from the &SessionInformation rowset.
--------------------	---

Returns

A boolean value: *True* if the row was successfully deleted, *False* if it wasn't.

Establish the Merchant Account

Establish with the merchant what services will be made available, what terms will apply, how you'll connect to the merchant's server, and how you'll present the services to your users. The resulting MIP, of course, will depend on your company's policies and procedures, and will vary considerably from merchant to merchant. Here are some suggested points to cover:

1. Meet with merchant representatives to evaluate the business and technical feasibility of an MIP.
2. Develop a checklist or questionnaire which covers the technical, functional and administrative aspects of the proposed MIP, and submit it to the merchant. It should serve to make both you and the merchant more aware of the parameters necessary for a successful integration. Some possible questions:
 - Will the merchant accept XML formatted data? If so, you can use the HTTPEnable.dll run-time plug-in.
 - Does the merchant require a merchant user ID for sign-on? If so, what format must it have?
 - What other input does the merchant require (company ID, password and user ID are handled by the Single Sign-On Framework)?
 - What output does the merchant produce, and what form does it take (machine name and session ID are handled by the Single Sign-On Framework)?
3. Determine what proprietary APIs the merchant provides, if any.
4. Determine whether the merchant services are best implemented as hidden application functionality, branded application functionality, or merchant HTML content.

Create the MIP

The MIP you develop will use the following elements:

- A business interlink design time plug-in.
- A business interlink run-time plug-in.
- A business interlink definition.
- An application message definition (in some cases).
- Merchant categories (in some cases).
- A merchant profile.
- One or more application pages or page controls.
- One or more PeopleCode programs.

In the following sections, we describe each of the development tasks necessary for creating a typical MIP.



You'll need to collect some of the profile information from the merchant. Read through the following sections to determine what information is necessary, so you can have it ready before you start.

Create the Business Interlink

The HTTPEnable.dll business interlink is the preferred vehicle for interacting with any merchant. Your merchant may require a custom run-time plug-in for sign-on.

Create the design time plug-in—see PeopleSoft Business Interlink Application Developer Guide.

Create a custom run-time plug-in if the merchant requires it—see PeopleSoft Business Interlink Runtime Plug-in Programming Guide.

Create the business interlink object definition using the design time plug-in—see Designing a Business Interlink Definition.

Create Merchant Categories

A merchant who provides multiple services may supply a different URL for each service. This is where merchant categories come into play. You'll assign each service to a different category, enabling you to record a different URL for the merchant under each category. You can then develop one or more MIPs for that merchant using a single profile; your PeopleCode for each MIP should select the appropriate URL based on category. You can use any of the categories delivered with your PeopleSoft application, or you can create your own.



Merchant categories have no intrinsic meaning; they simply provide a way to group MIPs by function. You can associate any category with a given URL, but for maintenance reasons it's helpful to have a category name and description that match the URL's purpose.

Follow this procedure to create any new categories you need, then apply the categories when you modify or create the merchant profile.

To create a new merchant category:

1. Select Go, PeopleTools, Utilities, Use, Merchant Categories, then Add a new record.

The new Merchant Category ID you enter must be upper case and no longer than 10 characters. The Merchant Categories page will appear.
2. Set the Status to *Active* for the current effective-dated row.
3. Enter a Description and Short Description of the category for the current effective-dated row.
4. Click Save to save the new merchant category.



For more information about merchant categories, see Merchant Categories Component.

Create a New Merchant Profile

The Single Sign-On Framework requires a merchant profile in order to implement the MIP. The merchant profile stores and maintains all the information and parameters needed for managing your relationship with the merchant, including integrating the merchant services with your application, maintaining account security, applying effective dating, conveniently updating connection and sign-on parameters, applying alternate Web site URLs, and invoking custom attributes. The PeopleCode programs you create for the MIP will use the information in the profile to configure and manage its interactions with the merchant.

To create a new merchant profile:

1. Select Go, PeopleTools, Utilities, Use, Merchant Profile, Merchant Profile, then Add a new record.

The new merchant ID you enter must be upper case and no longer than 30 characters.
2. On the Merchant Profile page, select the Merchant Enabled check box.

MIPs using this profile won't retrieve authentication parameters unless this box is selected.
3. Enter a Description and Short Description of the merchant.
4. Enter contact information in the Relationship Manager, Relationship Manager Email, and

Relationship Manager Phone fields.

Get this information from your merchant; it's for your convenience. The relationship manager is the merchant's support representative.



If the merchant has a different relationship manager for each service it provides, you may want to create a separate profile for each service.

5. Click Save to save the new profile.



For more information about the Merchant Profile page, see Merchant Profile Page.

Set Up Connection and Authentication Parameters

This information is needed by the Single Sign-On Framework to connect and establish a session on the merchant's Web site. If the Merchant Authentication parameters vary depending on the service provided, you'll need to create a separate profile for each service.

To set up connection and authentication parameters:

1. On the Merchant Authentication page, select the Merchant User ID Type expected by the merchant Web site.

Obtain this information from your merchant. This selection will have no effect if the merchant doesn't require a merchant user ID.



If the merchant expects a Merchant User ID Type that can't be generated by one of the available choices, select an available choice for now. You'll use PeopleCode to establish the correct Merchant User ID during a later procedure.

2. Enter the Company ID and Company Authentication Token that the MIP will use for sign-on.

This is your company's sign-on password; obtain it from your merchant. When you tab out of the Company Authentication Token edit box, a Confirm edit box will appear.

3. Retype the Company Authentication Token in the Confirm edit box.
4. Enter the complete HTTP URL that the MIP will use as the default initial point of contact with the merchant Web site.

This will probably be the sign-on URL—obtain this information from your merchant.

5. Click Save to save the modified profile.



For more information about the Merchant Authentication page, see Merchant Authentication Page.

Specify Business Interlink Information

Although you don't need to enter anything on the Merchant BI Overrides page to develop a working MIP, configuring its business interlinks on this page is advisable for several reasons:

- You can manage your MIP and its constituent elements through a single component.
- You can override the default settings for a business interlink, and you can change the overrides at any time without altering your PeopleCode.

Follow this procedure for each business interlink that will be used by the MIP.

To specify a business interlink:

1. On the Merchant BI Overrides page, enter the name of the Business Interlink.

Use the Business Interlink edit box prompt button to select a business interlink definition from the list of available definitions. The business interlink's parameter names and default values will appear in the Business Interlink Settings grid.

2. Select the Sign-On Business Interlink check box if the displayed business interlink will be used to sign on to the merchant Web site.

At most, one business interlink can be designated as the Sign-On Business Interlink for your MIP.

3. Enter a new value for each Parameter Value you wish to override.

If your MIP requires parameter values different from the business interlink's default settings, the Business Interlink Settings grid enables you to specify those values without interfering with the defaults.

4. Select the Override check box for each parameter you wish to override.
5. Clear the Override check box for each parameter you don't wish to override.
6. Click Save to save the modified profile.

Only the grid rows with a selected Override check box will remain on the page, and the MIP will use their parameter values in place of the business interlink's default settings.



For more information about the Merchant BI Overrides page, see Merchant BI Overrides Page.

Specify Alternate Merchant URLs by Category

Follow this procedure if your MIP will connect to the merchant at a different Web site URL for each transaction. Follow the procedure once for each URL.

To specify a merchant category and URL:

1. On the Merchant Category page, select a category you want to use.
2. The category name and description should reflect the types of transaction they represent.
3. Enter the complete Merchant Website URL to associate with the selected category.
4. Click Save to save the modified profile.



For more information about the Merchant Category page, see Merchant Category Page.

Specify Application Attributes

Adding information to the Application Attributes page is necessary only if you want to apply extra criteria to your MIP that you can use for your own purposes. These criteria are stored as name/value pairs.

To specify an application attribute:

1. On the Application Attributes page, if an empty row isn't available in the Application Attributes grid, click the Plus button to add a new row.
2. Enter a new uppercase Attribute Name.

The attribute name should reflect type of information it represents.
3. Enter the uppercase Attribute Value you wish to associate with the Attribute Name you entered.
4. Click Save to save the modified profile.



For more information about the Application Attributes page, see Application Attributes Page.

Build or Modify the Application Page



For more information, see Application Designer.

Create the MIP PeopleCode

The exact composition of your MIP PeopleCode will depend on:

- The type of sign-on and account terms required by the merchant.
- The business interlinks your MIP uses.
- The number of different transactions your MIP conducts.
- The level of session management your MIP requires.
- How your users will interact with your MIP.

The following examples demonstrate many of the basic features you'll include in your merchant integration PeopleCode programs, with an emphasis on the sign-on task. Non-sign-on transactions may use a different business interlink definition, design time plug-in, or run-time plug-in, and their PeopleCode won't include any of the sign-on and authentication elements.



The code fragments in this section may not comprise a complete MIP PeopleCode program, and code examples aren't necessarily in the order in which you'll want to use them in your program.

Declare the Merchant Single Sign-On Functions

```

Declare Function CreateSessionInformation PeopleCode
PSFUNCLIB_PTIC.MRCHSIGNONFNCTN FieldFormula;

Declare Function GetAuthenticationParms PeopleCode
PSFUNCLIB_PTIC.MRCHSIGNONFNCTN FieldFormula;

Declare Function InsertSessionData PeopleCode PSFUNCLIB_PTIC.MRCHSIGNONFNCTN
FieldFormula;

Declare Function GetSessionData PeopleCode PSFUNCLIB_PTIC.MRCHSIGNONFNCTN
FieldFormula;

Declare Function DeleteSessionData PeopleCode PSFUNCLIB_PTIC.MRCHSIGNONFNCTN
FieldFormula;

```



For more information about these functions, see Single Sign-On PeopleCode Functions.

Declare Data Objects

The global rowset object must be called &SessionInformation:

```
/* Global Rowset Object */
```

```

Global Rowset &SessionInformation;

/* Local Record Objects */

Local Record &AuthParms;

Local Record &SessionDataInsrt;

Local Record &SessionDataGet;

```

Define the Sign-On Function

A sign-on function is not required, but the following example code shows how to use the Single Sign-On functions and data structures. The details of this function depend very much on the specifics of your MIP and the requirements imposed by the merchant. It calls a sign-on business interlink, which could be a custom business interlink, or a business interlink based on HTTPEnable.dll, which is recommended. The function's parameters are the return values from the GetAuthenticationParms function, and it returns a record containing session information received from the merchant system.



You can use some parts of this sign-on function as the basis for PeopleCode for other transactions.

```

Function SignOn(&MerchantID, &MrchUserID, &CompanyID, &Token, &SignOnURL,
&MerchantURL) Returns Record

```

```

&ReturnValue = CreateRecord(Record.PSSESSIONDATA);

&ReturnValue.MERCHANTID.Value = &MerchantID;

```

Instantiate and configure the business interlink object

If the parameters passed to this function are the values returned from the GetAuthenticationParms function, the name of the business interlink you provide here must match the one selected as the Sign-On Business Interlink in the merchant profile:

```
&SIGNON_BI = GetInterlink(Interlink.MYSIGNONBI);
```

Because the GetAuthenticationParms function automatically returns any override value for the URL location of the sign-on business interlink, you can assign that value (if it exists) from the &SignOnURL parameter passed to this function:

```
&SIGNON_BI.URL = &SignOnURL;
```

GetAuthenticationParms also returns the value of the Merchant Web site URL field from the Merchant Authentication page, so if the business interlink has a MerchantURL parameter (as

those used for delivered MIPs do), you can override its default value by assigning the value from the &MerchantURL parameter passed to this function:

```
&SIGNON_BI.MerchantURL = &MerchantURL;
```

If your MIP needs to temporarily override the business interlink's default settings, you can retrieve those settings from the merchant profile's PSMERCHBIPARMS record and apply them to the business interlink object. Here we apply the override value for the User_DN parameter:

```
&BName = "MYSIGNONBI";

&ParameterName = "User_DN";

SQLExec("SELECT IOVALUE FROM PSMERCHBIPARMS WHERE EFFDT = (SELECT
MAX(A.EFFDT) FROM PSMERCHBI A WHERE A.MERCHANTID=:4 AND A.EFFDT <= %DATEIN(:1))
AND IONAME = :2 AND IOSETTINGNAME = :3", %Date, &BName, &ParameterName,
&MerchantID, &OverrideValue);

&SIGNON_BI.User_DN = &OverrideValue;
```

If your MIP connects to different Web site URLs depending on merchant category, you can retrieve the appropriate URL for a category from the merchant profile's PSMERCHANTCAT record and use it to override the business interlink's default merchant Web site URL:

```
&MerchantCat = "Category01";

SQLExec("SELECT MERCHWSITEURL FROM PSMERCHANTCAT WHERE EFFDT = (SELECT
MAX(A.EFFDT) FROM PSMERCHANTCAT A WHERE A.MERCHANTID=:2 AND A.EFFDT <=
%DATEIN(:1)) AND MERCHANTID = :2 AND MERCHANTCAT = :3", %Date, &MerchantID,
&MerchantCat, &MerchCatURL);

&SIGNON_BI.MerchantURL = &MerchCatURL;
```

Sign on and obtain session information

UserID, GroupID and AuthToken are the input parameters required by this merchant for sign-on, so assign them the values of the sign-on function's parameters:

```
&SIGNON_BI.AddInputRow("UserID", &MrchUserID, "GroupID", &CompanyID,
"AuthToken", &Token);

&EXECSLT = &SIGNON_BI.Execute();

If (&EXECSLT <> 1) Then

    /* Signon failed; respond accordingly */

Else

    &RSLT = True;
```

Session_ID and Machine_Name are the session parameters returned by this merchant. The Single Sign-On Framework currently supports only these two fields:

```
While &RSLT
```

```

        &RSLT = &SIGNON_BI.FetchNextRow("Session_ID", &SESSION, "Machine_Name",
&MACHINENAME, "return_status", &ReturnStatus, "return_status_msg",
&ReturnStatusMsg);

    End-While;

End-If;

&ReturnValue.MRCHSESSION.Value = &SESSION;

&ReturnValue.MRCHMACHINENAME.Value = &MACHINENAME;

Return &ReturnValue;

End-Function;

```



The inputs you provide and the outputs returned by the merchant are determined entirely by agreement between you and the merchant.

Create and Initialize Data Objects and Variables

You'll know the merchant ID at application design time; &AuthParms holds authentication information:

```

/* Use this &MERCHANTID value for all transactions */

&MERCHANTID = "MYMERCHANT";

&AuthParms = CreateRecord(Record.PSAUTHPARMS);

```

Create the global rowset object &SessionInformation, based on the PSSESSIONDATA record definition. &SessionDataInsrt holds session information returned by the sign-on function; &SessionDataGet holds data retrieved from &SessionInformation:

```

CreateSessionInformation();

/* Use variables based on PSSESSIONDATA for all
 * transactions if you maintain session information */

&SessionDataInsrt = CreateRecord(Record.PSSESSIONDATA);

&SessionDataGet = CreateRecord(Record.PSSESSIONDATA);

```

GetAuthenticationParms retrieves authentication information from the Merchant Profile, including the URL locations of the sign-on business interlink and the merchant Web site:

```

&AuthParms = GetAuthenticationParms(&MERCHANTID);

```



```

If &AuthParms = Null Then

    /* Application specific error handling */

End-If;

```

Sign On and Manage Session Information

Submit the Merchant User ID, Company ID and decrypted authentication token:

```

&SessionDataInsrt = SignOn(&AuthParms.MERCHANTID.Value,
&AuthParms.MRCHUSERID.Value, &AuthParms.COMPANYID.Value,
&AuthParms.WRKTOKEN.Value, &AuthParms.SIGNONBIURL.Value,
&AuthParms.MERCHWSITEURL.Value);

```

Insert the returned session data into the &SessionInformation object:

```

If Not InsertSessionData(&SessionDataInsrt) Then

    /* Application specific error handling */

End-If;

```

Retrieve session data from the &SessionInformation object, based on merchant ID:

```

/* Use &SessionInformation to determine the MRCHSESSION *
* and MRCHMACHINENAME values for all transactions */
&SessionDataGet = GetSessionData(&MERCHANTID);

If &SessionDataGet = Null Then

    /* Application specific error handling */

End-If;

```

You may want to eliminate information about a merchant session the current user has completed—delete a session row from the &SessionInformation object, based on merchant ID:

```

If Not DeleteSessionData(&MERCHANTID) Then

    /* Application specific error handling */

End-If;

```

Use Application Attributes

One possible use for application attributes is to store the meanings of arbitrarily named data items used by the merchant so they can be referenced with more understandable names that you assign to them. For example, if the “application developer” skill is represented on the merchant site by the code SKILLRECRUIT02, you could store the code as the more understandable application attribute APPDEV. This example retrieves the code so you can submit it to the merchant for a search:

```

/* Retrieve the value of the APPDEV attribute to use */

* as the business interlink's SkillCode parameter */

&AppAttrName = "APPDEV";

SQLExec("SELECT MRCHAPPATTRVALUE FROM PSMERCHANTAPP WHERE EFFDT = (SELECT
MAX(A.EFFDT) FROM PSMERCHANTAPP A WHERE A.MERCHANTID=:2 AND A.EFFDT <=
%DATEIN(:1)) AND MERCHANTID = :2 AND MRCHAPPATTRNAME = :3", %Date, &MerchantID,
&AppAttrName, &AppAttrValue);

&SIGNON_BI.SkillCode = &AppAttrValue;

```

Implementing an MIP

In this section, we assume you've determined which merchant services you want to make available to users of your PeopleSoft application, and we describe how to implement the Merchant Integration Points that provide those services.

Depending on the MIP you're implementing, you may also need to be familiar with PeopleSoft Security.

Implementation Tasks



If you developed a custom MIP for your PeopleSoft application, you may have already completed one or more of the tasks listed below. All three tasks are necessary, so you can use this section as a checklist for your custom MIP.

Implementing an MIP involves the following tasks:

1. Activate the merchant account—establish an account for your organization with the participating merchant whose services you want to use.
2. Configure the Merchant Profile—use the appropriate Merchant Profile pages to configure and enable the MIP that provides access to the merchant services.
3. Manage user access to merchant services—configure your application's security to provide appropriate user access to the part of your application that uses the MIP.

When you complete all implementation tasks, your authorized users will be able to use the MIP.

Implementing MIPs Delivered with Your PeopleSoft Application

MIPs delivered with PeopleSoft applications vary widely with respect to the way you should complete the implementation. An MIP with a simple purpose, minimal security concerns, or a highly automated setup procedure may require little fuss on your part to become operational. An

MIP with high security requirements or a wide range of possible configurations may involve contacting the merchant, modifying the merchant profile, or restricting user access to certain application pages.

Your application documentation is your primary source of information about implementing its MIPs. You may not need to use this chapter at all, so refer to your application PeopleBook first.

Activate the Merchant Account

The procedure for activating your account varies from merchant to merchant, and may be required before you configure the MIP, after you enable the MIP, or as part of the initial connection process with the merchant. See your application documentation for details about the relevant MIP before proceeding.



If you're implementing an MIP you developed to enhance your PeopleSoft application, you should have already established and activated a merchant account as part of that process.

When you activate your account, you may receive any of the following:

- A company ID and authentication token (password) for signing on to the merchant Web site.
- Information about the merchant's relationship manager (account contact).
- A Web site URL the MIP should use to connect to the merchant.

This information should be added to the MIP's merchant profile.



For more information about adding account information to the MIP, see [Configure the Merchant Profile](#).

Configure the Merchant Profile

When you install your PeopleSoft application, all the elements that support the MIPs delivered with the application are deployed and ready for use, including business interlink or application message definitions, business interlink plug-ins, page controls, a merchant profile, and PeopleCode that manages those elements.



Your application may require other configuration steps for this MIP. Refer to your application documentation for details.

PeopleSoft's MIPs are delivered with generic connection, security and account information populating their merchant profiles. This information may not be accurate with respect to your company, so you must update it for the accounts you set up. In most cases, configuring an MIP

requires only that you modify the MIP's merchant profile with information specific to your company. The modifications required for some MIPs may be minor, while for others they may be major. You may also need to enter some of the information before establishing an account with the merchant, then add the rest after establishing the account.

Because the merchant profile is effective-dated, you should add a new row to the Merchant Profile component, which will acquire the values of the displayed row. You can then modify the new row with your company's information.



Refer to your application documentation to determine which modifications to make and when. The modifications may include any of the following items in the Merchant Profile component.

Modifying the Merchant Profile Page

To modify a merchant profile:

1. Select Go, PeopleTools, Utilities, Use, Merchant Profile, Merchant Profile, then open the merchant profile.
2. Create a new effective-dated row if necessary.
3. Select the Merchant Enabled check box.
4. Enter the relationship manager's name, email address and telephone number.
5. This information is for your reference; the MIP will work without it.
6. Click Save to save the modified profile.



For more information about the Merchant Profile page, see Merchant Profile Page.

Modifying the Merchant Authentication Page

To modify merchant authentication parameters:

1. Select Go, PeopleTools, Utilities, Use, Merchant Profile, Merchant Authentication, then open the merchant profile.
2. Select the appropriate User ID Type for the MIP.

This may already be correctly set; see your application documentation for more information.
3. Enter the Company ID and Company Authentication Token that the MIP will use for sign-on.

This is your company's sign-on password; obtain it from your merchant. When you tab out of the Company Authentication Token edit box, a Confirm edit box will appear.

4. Retype the Company Authentication Token in the Confirm edit box.
5. Enter the Merchant Website URL to which the MIP should connect.

Depending on the scope and terms of your account, the merchant may want to direct your users to a URL specifically designed for your company. This may already be correctly set; see your application documentation for more information.

6. Click Save to save the modified profile.



For more information about the Merchant Authentication page, see Merchant Authentication Page.

Modifying the Merchant BI Overrides Page

Usually, this page requires no modification when implementing an MIP.



For more information about the Merchant BI Overrides page, see Merchant BI Overrides Page.

Modifying the Merchant Category Page

The MIP may be designed to connect to a separate Website URL for each category of transaction. Refer to your application documentation; if it indicates that the MIP uses such separate URLs, and the merchant has provided you with such URLs specifically designed for your company, follow this procedure for each listed category.

To modify merchant categories:

1. Select Go, PeopleTools, Utilities, Use, Merchant Profile, Merchant Category, then open the merchant profile.
2. Select a merchant category row.
3. Enter the correct Merchant Website URL for the displayed merchant category.
4. Click Save to save the modified profile.



For more information about the Merchant Category page, see Merchant Category Page.

Modifying the Application Attributes Page

Usually, this page requires no modification when implementing an MIP.



For more information about the Application Attributes page, see Application Attributes Page.

Manage User Access to Merchant Services

Merchant integration is achieved with an authorized merchant account accessed through an application page. The users don't have to know any merchant account IDs or passwords, therefore you're requiring only a single PeopleSoft sign-on by each user.

The set of available merchant integrations can vary from user to user at the same site in effect, but not in fact. When you enable an MIP, it's available to all of your PeopleSoft users, but you can control access to merchant services in two ways:

- Using PeopleSoft navigation restrictions.

Your PeopleSoft application applies its security settings to your users' PeopleSoft roles, which determine their access to specific application pages. You can effectively use roles to control access to the merchant by controlling access to application pages containing the relevant merchant links and content. Users without access won't be able to navigate to those pages. This is a standard PeopleSoft security feature.



For more information about controlling user access to pages, see Security.

- Using the merchant's account access.

If a component or page provides other functionality in addition to a merchant service, you may wish to give some users access to the non-merchant functionality, but not to the merchant services. Most merchants should be able to provide access to only those users you specify, based on the merchant user ID type defined in the merchant profile. An unauthorized user can trigger a sign-on to the merchant site, but can't use the merchant's services. Refer to your application documentation and the merchant to determine whether you can manage user access in this way.

CHAPTER 3

The PeopleSoft API Repository

The PeopleSoft API Repository allows PeopleCode and third party integrators to discover the internally available classes, methods, and properties provided by PeopleSoft for integration. The Repository is useful to third-party integrators who integrate in a generic fashion: middleware providers, testing tool providers and automated documentation providers.

The PeopleSoft API Repository is not a necessary interface for integrators who integrate at the business rule level, such as integration with a Expense Report, and so on. Those integrators should use Component Interfaces or Business Interlinks.

Using the Repository

The major job of the Repository is to describe the available PeopleSoft APIs. It provides a number of mechanisms to determine the classes available in the API, the properties of each class, the methods of a class (along with the required parameters), and information concerning which *group* a class belongs (known as a Namespace).

The process of determining the information about the API is known as *discovery*. Third-party integrators may use information found through discovery to drive generic integration tools.

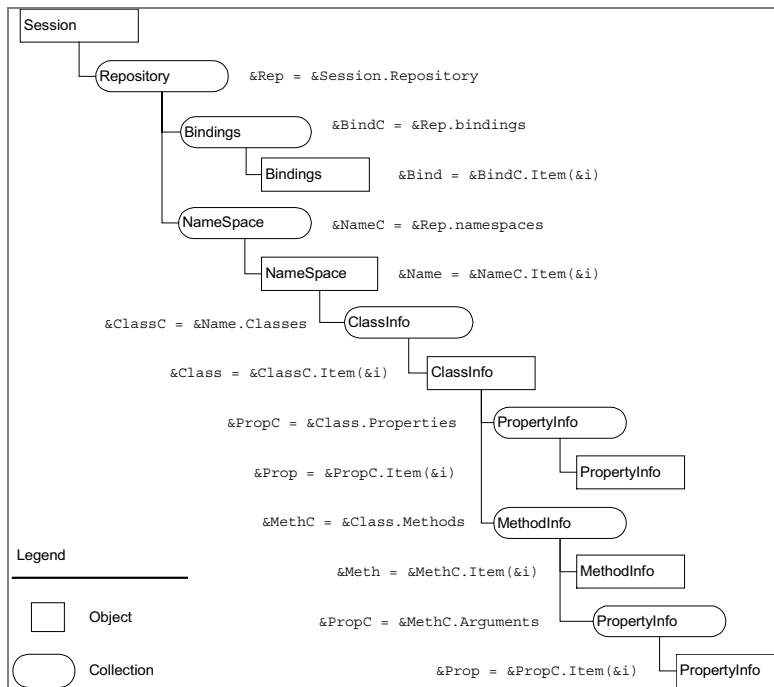
The Repository is divided into Namespaces. Each Namespace contains a collection of related classes. Example Namespaces include "PeopleSoft", "ComponentInterface", "Trees", or "BusinessInterlinks".

A **class** defines a related set of methods and properties. Using the Repository, you can determine the methods and properties that are available, and can be used on any object returned by a call to the PeopleSoft API. An **instance** of a class is known as an **object**.

A **property** is a data item of an object that has both a Name and Type (string, number, boolean, etc). Some properties are used for inputting data to a class, some are used for getting data from a class, some are used for both. Whether a property is used for input or output or both is known as **Usage**.

A **method** is a function you can call on an object. Methods have a Name and a return Type (string, number, boolean, etc). Methods also have a collection of arguments that must be set prior to invoking the method. Methods arguments have identical attributes to properties.

The following flow chart shows the different types of objects and collections instantiated from the Repository:



Flow chart for Repository

Example of Using the Repository

The following example gets information for the class ABS_HIST from the Namespace Component Interface, and writes it to the file BC.TXT.

The following is the complete code sample, followed by the flat file. After that are steps that explain each line.

Example Using Visual Basic contains an exact copy of this example, done in Visual Basic.

```

Local ApiObject &MYSESSION;

Local ApiObject &MYCI;

Local string &OutTEXT;

Local File &MYFILE;

&MYSESSION = GetSession();

&MYSESSION.Connect(1, "EXISTING", "", "", 0);

&MYFILE = GetFile("CI.txt", "A");

```



```

&NAMESPACES = &MYSESSION.Repository.Namespaces;

&NAMESPACE = &NAMESPACES.ItemByName("CompIntfc");

&OutTEXT = "Namespace = " | &NAMESPACE.Name;

&MYFILE.WriteLine(&OutTEXT);

&CLASSES = &NAMESPACE.classes;

&CLASS = &CLASSES.ItemByName("ABS_HIST");

&OutTEXT = "  Class: " | &CLASS.Name;

&MYFILE.WriteLine(&OutTEXT);

&OutTEXT = "      Methods";

&MYFILE.WriteLine(&OutTEXT);

&METHODS = &CLASS.methods;

For &K = 0 To &METHODS.Count - 1

    &METHOD = &METHODS.item(&K);

    &OutTEXT = "          " | &METHOD.name | ": " | &METHOD.Type;

    &MYFILE.WriteLine(&OutTEXT);

    &ARGUMENTS = &METHOD.arguments;

    For &M = 0 To &ARGUMENTS.count - 1

        &ARGUMENT = &ARGUMENTS.item(&M);

        &OutTEXT = "          " | &ARGUMENT.name | ": " | &ARGUMENT.type;

        &MYFILE.WriteLine(&OutTEXT);

    End-For;

End-For;

&OutTEXT = "      Properties";

```

```

&MYFILE.WriteLine(&OutTEXT);

&PROPERTIES = &CLASS.properties;

For &I = 0 To &PROPERTIES.count - 1

    &PROPERTY = &PROPERTIES.item(&I);

    &OutTEXT = "                " | &PROPERTY.name | ": " | &PROPERTY.type;

    &MYFILE.WriteLine(&OutTEXT);

End-For;

&MYFILE.Close();

```

The above code produces the following flat file:

```

Namespace = CompIntfc

Class: ABS_HIST

Methods

    Get: Boolean

    Save: Boolean

    Cancel: Boolean

    Find: ABS_HIST

    GetPropertyByName: Variant

Name: String

    SetPropertyByName: Number

Name: String

Value: Variant

    GetPropertyInfoByName: CompIntfcPropertyInfo

Name: String

Properties

    EMPLID: String

    LAST_NAME_SRCH: String

    NAME: String

    ABSENCE_HIST: ABS_HIST_ABSENCE_HISTCollection

```

```

interactiveMode: Boolean

getHistoryItems: Boolean

componentName: String

compIntfcName: String

stopOnFirstError: Boolean

propertyInfoCollection: CompIntfcPropertyInfoCollection

createKeyInfoCollection: CompIntfcPropertyInfoCollection

getKeyInfoCollection: CompIntfcPropertyInfoCollection

findKeyInfoCollection: CompIntfcPropertyInfoCollection

```

The following steps go through the code example line by line:

1. Get a session object.

Before you can access the API Repository, you have to get a session object. The session controls access to PeopleSoft, provides error tracing, allows you to set the runtime environment, and so on.

```

&MYSESSION = GetSession();

&MYSESSION.Connect(1, "EXISTING", "", "", 0);

```

2. Open the file

As this text is going to be written to a flat file, the next step is to open the file. If the file is already created, the new text will be appended to the end of it. If the file hasn't been created, the GetFile built-in function will create the file.

```

&MYFILE = GetFile("CI.txt", "A");

```

3. Get the Namespace you want.

Use the Namespaces property on the Repository object to get a collection of all the available Namespaces. We want to discover information about a Component Interface, so we specify CompIntfc in the ItemByName method to get that Namespace. With ItemByName, you must specify a Namespace that already exists. You'll receive a runtime error if you specify one that doesn't exist.

```

&NAMESPACES = &MYSESSION.Repository.Namespaces;

&NAMESPACE = &NAMESPACES.ItemByName("CompIntfc");

```

4. Write the text to the file.

Because all the information discovered is being written to a file, the next step is to write text to the file. This code writes the string Namespace, followed by the name of the namespace, to the file.

```
&OutTEXT = "Namespace = " | &NAMESPACE.Name;

&MYFILE.WriteLine(&OutTEXT);
```

5. Get the class you want, and write text to the file.

Use the Classes property on the Namespace object to get a collection of all the available classes. We want to discover information about the Component Interface named ABS_HIST, so we specify that using ItemByName. Then we write that information to the file.

```
&CLASSES = &NAMESPACE.classes;

&CLASS = &CLASSES.ItemByName("ABS_HIST");
```

```
&OutTEXT = " Class: " | &CLASS.Name;

&MYFILE.WriteLine(&OutTEXT);
```

6. Get the methods, arguments, and write the information to the file.

Use the Methods property on the Class object to get a collection of all the available methods. After you get each method, and write the information to the file, loop through and find all of the arguments for the method, then write that information to the file.

```
&OutTEXT = "      Methods";

&MYFILE.WriteLine(&OutTEXT);

&METHODS = &CLASS.methods;

For &K = 0 To &METHODS.Count - 1

    &METHOD = &METHODS.item(&K);

    &OutTEXT = "          " | &METHOD.name | ": " | &METHOD.Type;

    &MYFILE.WriteLine(&OutTEXT);

    &ARGUMENTS = &METHOD.arguments;

    For &M = 0 To &ARGUMENTS.count - 1

        &ARGUMENT = &ARGUMENTS.item(&M);

        &OutTEXT = "          " | &ARGUMENT.name | ": " | &ARGUMENT.type;

        &MYFILE.WriteLine(&OutTEXT);

    End-For;

End-For;
```

7. Get the properties and write the information to the file.

Use the **Properties** property on the **Class** object to get a collection of all the available properties. Write each property, with its type, to the file. At the end of the program, close the file.

```
&OutTEXT = "      Properties";

&MYFILE.WriteLine(&OutTEXT);

&PROPERTIES = &CLASS.properties;

For &I = 0 To &PROPERTIES.count - 1

    &PROPERTY = &PROPERTIES.item(&I);

    &OutTEXT = "          " | &PROPERTY.name | ": " | &PROPERTY.type;

    &MYFILE.WriteLine(&OutTEXT);

End-For;

&MYFILE.Close();
```

Repository Properties

Bindings

The **Bindings** property returns a reference to a **Bindings** collection.

This property is read-only.

Namespaces

The **Namespaces** property returns a reference to a **Namespaces** collection.

This property is read-only.

Bindings Collection Properties

Count

This property returns the number of Bindings Properties objects in the Bindings collection object.



Note. All repository counts begin at **zero**, not one.

This property is read-only.

Example

```
&COUNT = &BINDINGS.Count ;
```

Bindings Collection Methods

Item

Syntax

```
Item(number)
```

Description

The **Item** method returns a Bindings object that exists at the *number* position in the Bindings collection executing the method

Parameters

<i>number</i>	Specify the position number in the collection of the Bindings object that you want returned.
---------------	--

Returns

A reference to a Bindings object, NULL otherwise.

Example

```
For &N = 0 to &BINDINGS.Count - 1  
  
    &BINDING = &BINDINGS.Item(&N) ;  
  
    /* do processing */  
  
End-For;
```

Bindings Properties

Name

This property returns the name of the object as a string.

This property is read-only.

Bindings Methods

Generate

Syntax

```
Generate()
```

Description

This method is a reserved internal function, and shouldn't be used at this time.

Namespaces Collection Properties

Count

This property returns the number of Namespaces Properties objects in the Namespaces collection object.



Note. All repository counts begin at **zero**, not one.

This property is read-only.

Example

```
&COUNT = &NameC.Count;
```

Namespaces Collections Methods

Item

Syntax

```
Item(number)
```

Description

The **Item** method returns a Namespaces object that exists at the *number* position in the Namespaces collection executing the method

Parameters

number Specify the position number in the collection of the Namespaces object that you want returned.

Returns

A reference to a Namespaces object, NULL otherwise.

Example

```
For &N = 0 to &NAMESPACES.Count - 1  
  
    &NAMESPACE = &NAMESPACES.Item(&N);  
  
    /* do processing */  
  
End-For;
```

ItemByName

Syntax

```
ItemByName(name)
```

Description

The **ItemByName** method returns the item specified by *name*. *name* is case insensitive.

Parameters

name Specify the name of the Namespaces object that you want returned. This parameter takes a string value.

Returns

A reference to a Namespaces object, NULL otherwise.

Example

```
&NAMESPACE = &NAMESPACES.ItemByName("BusinessComponent");
```

Namespaces Properties

Classes

This property returns a reference to a ClassInfo Collection Properties collection.

This property is read-only.

Example

```
&CLASSC = &NAME.Classes;
```

Name

This property returns the name of the object as a string.

This property is read-only.

Namespaces Methods

CreateObject

Syntax

```
CreateObject(classname)
```

Description

This method is a reserved internal function and shouldn't be used at this time.

ClassInfo Collection Properties

Count

This property returns the number of ClassInfo Properties objects in the ClassInfo collection object.



Note. All repository counts begin at **zero**, not one.

This property is read-only.

Example

```
&COUNT = &InfoC.Count;
```

ClassInfo Collection Methods

Item

Syntax

Item (*number*)

Description

The **Item** method returns a ClassInfo object that exists at the *number* position in the ClassInfo collection executing the method

Parameters

<i>number</i>	Specify the position number in the collection of the ClassInfo object that you want returned.
---------------	---

Returns

A reference to a ClassInfo object, NULL otherwise.

Example

```
For &N = 0 to &CLASSES.Count - 1  
  
    &CLASS = &CLASSES.Item(&N);  
  
    /* do processing */  
  
End-For;
```

ItemByName

Syntax

ItemByName (*name*)

Description

The **ItemByName** method returns the item specified by *name*. *name* is case insensitive.

Parameters

<i>name</i>	Specify the name of the ClassInfo object that you want returned. This parameter takes a string value.
-------------	---

Returns

A reference to a ClassInfo object, NULL otherwise.

Example

```
&CLASS = &CLASSES.ItemByName ("ABS_HIST") ;
```

ClassInfo Properties

Documentation

This property returns a description of the class, as a string. This doesn't actually return all the documentation for the class, just a brief description.

This property is read-only.

Methods

This property returns a reference to a MethodInfo Collection Methods collection.

This property is read-only.

Name

This property returns the name of the object as a string.

This property is read-only.

Properties

This property returns a reference to a PropertyInfo Collection Methods collection.

This property is read-only.

MethodInfo Collection Methods

Item

Syntax

```
Item(number)
```

Description

The **Item** method returns a MethodInfo object that exists at the *number* position in the MethodInfo collection executing the method

Parameters

number Specify the position number in the collection of the MethodInfo object that you want returned.

Returns

A reference to a MethodInfo object, NULL otherwise.

Example

```
For &K = 0 To &METHODS.Count - 1

    &METHOD = &METHODS.item(&K);

    &OutTEXT = "          " | &METHOD.name | ": " | &METHOD.Type;

    &MYFILE.WriteLine(&OutTEXT);

End-For;
```

ItemByName

Syntax

ItemByName (*name*)

Description

The **ItemByName** method returns the item specified by *name*. *name* is case insensitive.

Parameters

name Specify the name of the MethodInfo object that you want returned. This parameter takes a string value.

Returns

A reference to a MethodInfo object, NULL otherwise.

Example

```
&METHOD = &METHODS.ItemByName("Save");
```

MethodInfo Collection Properties

Count

This property returns the number of MethodInfo Properties objects in the MethodInfo collection object.



Note. All repository counts begin at **zero**, not one.

This property is read-only.

Example

```
&COUNT = &MethC.Count;
```

MethodInfo Properties

Arguments

This property returns a reference to a PropertyInfo Collection Methods collection.

This property is read-only.

Documentation

This property returns a description of the class, as a string. This doesn't actually return all the documentation for the class, just a brief description.

This property is read-only.

Name

This property returns the name of the object as a string.

This property is read-only.

Type

This property returns the type of the method. Valid values include:

- Bool
- Number
- Float
- String
- Variant
- Blob (Binary large object)
- any API class name

This property is read-only.

PropertyInfo Collection Methods

Item

Syntax

Item(*number*)

Description

The **Item** method returns a PropertyInfo object that exists at the *number* position in the PropertyInfo collection executing the method

Parameters

<i>number</i>	Specify the position number in the collection of the PropertyInfo object that you want returned.
---------------	--

Returns

A reference to a PropertyInfo object, NULL otherwise.

Example

```
For &K = 0 To &PROPERTIES.Count - 1

    &PROPERTY = &PROPERTIES.item(&K) ;

    &OutTEXT = "          " | &PROPERTY.name | ": " | &PROPERTY.Type;

    &MYFILE.WriteLine(&OutTEXT);

End-For;
```

ItemByName

Syntax

ItemByName(*name*)

Description

The **ItemByName** method returns the item specified by *name*. *name* is case insensitive.

Parameters

name Specify the name of the PropertyInfo object that you want returned. This parameter takes a string value.

Returns

A reference to a PropertyInfo object, NULL otherwise.

Example

```
&PROPERTY = &PROPERTIES.ItemByName("GetHistoryItems");
```

PropertyInfo Collection Properties

Count

This property returns the number of PropertyInfo Properties objects in the PropertyInfo collection object.



Note. All repository counts begin at **zero**, not one.

This property is read-only.

Example

```
&COUNT = &PropC.Count;
```

PropertyInfo Properties

Documentation

This property returns a description of the class, as a string. This doesn't actually return all the documentation for the class, just a brief description.

This property is read-only.

Name

This property returns the name of the object as a string.

This property is read-only.

Type

This property returns the data type. Valid values are:

- Bool
- Number
- Float
- String
- Variant
- Blob (Binary large object)
- or any API class name.

This property is read-only.

Usage

This property returns a number that describes which direction the specified property (or argument) can be passed. The valid values are:

<i>Value</i>	<i>Description</i>
0	Can be passed into PeopleSoft API
1	Can be passed out of PeopleSoft API
2	Can be passed either into or out of PeopleSoft API

This property is read-only.

Example Using Visual Basic

The following example gets information for the class ABS_HIST from the Namespace Component Interface.

```
Private Sub Command1_Click()

    '*****

    '* TacDemo: Example Repository Usage from Visual Basic

    '*

    '* Copyright (c) 1999 PeopleSoft, Inc. All rights reserved.

    '*****
```



```
' Declare variables

Dim oSession As New PeopleSoft_PeopleSoft.Session

Dim oPSMessages As PSMessageCollection

Dim oPSMessage As PSMessage

' Establish a PeopleSoft Session

nStatus = oSession.Connect(1, "//PSOFT0070698:9001", "PTDMO", "PTDMO", 0)

' Enable error-handler

On Error GoTo ErrorHandler

' Get a Component Interface "shell"

Dim oNamespaces As NamespaceCollection

Dim oNamespace As Namespace

Dim oClasses As ClassInfoCollection

Dim oClass As ClassInfo

Dim oMethods As MethodInfoCollection

Dim oMethod As MethodInfo

Dim oArguments As PropertyInfoCollection

Dim oArgument As PropertyInfo

Dim oProperties As PropertyInfoCollection

Dim oProperty As PropertyInfo

Set oNamespaces = oSession.Repository.namespaces

Set oNamespace = oNamespaces.ItemByName("ComponentInterface")

Dim outText As String

outText = "Namespace = " & oNamespace.Name & vbNewLine
```

```
Set oClasses = oNamespace.classes

Set oClass = oClasses.ItemByName("ABS_HIST")

outText = outText & "    Class: " & oClass.Name & vbNewLine

outText = outText & "        Methods" & vbNewLine

Set oMethods = oClass.methods
For k = 0 To oMethods.Count - 1
    Set oMethod = oMethods.Item(k)

    outText = outText & "            " & oMethod.Name & ": " & oMethod.Type
    & vbNewLine

    Set oArguments = oMethod.arguments
    For m = 0 To oArguments.Count - 1
        Set oArgument = oArguments.Item(m)

        outText = outText & "                " & oArgument.Name & ": " &
oArgument.Type & vbNewLine
    Next
Next

outText = outText & "        Properties" & vbNewLine

Set oProperties = oClass.properties
For k = 0 To oProperties.Count - 1
    Set oProperty = oProperties.Item(k)

    outText = outText & "            " & oProperty.Name & ": " &
oProperty.Type & vbNewLine
Next

txtResults = outText
```

```

' Leave before we encounter the error handler

Exit Sub

ErrorHandler:

    If Err.Number = 1001 Then                ' PeopleSoft Error

        Set oPSMessages = oSession.PSMessages

        If oPSMessages.Count > 0 Then

            For i = 1 To oPSMessages.Count

                Set oPSMessage = oPSMessages.Item(i)

                MsgBox (oPSMessage.Text)

            Next i

            oPSMessages.DeleteAll

        Else

            MsgBox ("PS Api Error. No additional information available from
Session log")

        End If

    Else                                    ' VB Error

        MsgBox ("VB Error: " & Err.Description)

    End If

End Sub

```

Summary of Repository Methods and Properties

This table contains a list of all the Repository objects plus their methods. Methods that can be used by a class are marked with an "X".

Method	Bindings coll	Namespaces collection	Classinfo collection	MethodInfo collection	Propertyinfo collection
CreateObject (<i>classname</i>)		X			

Method	Bindings coll	Namespaces collection	Classinfo collection	MethodInfo collection	Propertyinfo collection
Generate()	X				
Item(<i>number</i>)	X	X	X	X	X
ItemByName(<i>name</i>)		X	X	X	X

This table contains a list of all the Repository objects plus their properties. All properties are read-only.

Property	Rep coll/ object	Binding coll/ object	Namespace coll/ object	Classinfo coll/ object	MethodInfo coll/ object	PropertyInfo coll/ object
Arguments					RO	
Bindings	RO					
Classes			RO			
Count		RO	RO	RO	RO	RO
Documentation				RO	RO	RO
Generate		RO				
Methods				RO		
Name		RO	RO	RO	RO	RO
Namespaces	RO					
Properties				RO		
Type					RO	RO
Usage						RO

CHAPTER 4

Introducing File Layout

A **File Layout** is a definition (or mapping) of a file to be processed. It identifies where in a file data fields are located. Once a File Layout has been created, you can write PeopleCode programs that use the file object, which in turn use the File Layout, to either read data from or write data to a file.

You don't have to create a file layout for accessing data in a file. PeopleTools supports reading and writing to plain text files, as well as to files that have a format based on a File Layout.

- If the file is a plain text file, data is read or written using text strings.
- If the file is based on a File Layout, you can use text strings, rowset or record objects.

Using a File Layout greatly simplifies reading, writing and manipulating hierarchical transaction data with PeopleCode.



For more information about using PeopleCode to access a file, see File Class.

Fields: A Breakdown of Files

A file, in the simplest sense, is a collection of fields in a text format. In order to interface with each field within a file, the field must have a 'describable' location. The method of this 'description' falls into one of the following categories based on File Formats:

Fixed Positional	Each field has a starting position and a length which is their descriptive location.
Sequence Positional (CSV)	Fields are separated and the order of appearance is their descriptive location.
Tagged (XML)	Each field has predefined tags (or identifiers) surrounding it.



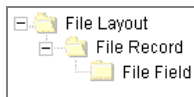
For more information about the different file formats, see Supported File Formats.

Relationships exist between fields, consequently, there must be a way to logically group fields into a collection. In the RDBMS world these collections are Tables or Records. Each line within a file can be considered a collection of fields.



Note. With some file formats, the logical concept of a line may actually span multiple physical lines, but the concept of collections of fields remains.

This gives us the following structure for a file:



File Layout Structure

where a File Layout is simply a collection of File Records, which in turn are a collection of Fields, each of which has a describable location.



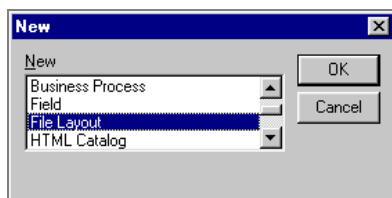
Note. To preclude confusion with the often used record and field names, we refer to the collection as a **File Record** and fields as **File Fields** when pertaining to a File Layout.

Creating a File Layout in Application Designer

To create a File Layout, you need to start Application Designer. The following example covers the easiest case: creating a File Layout from existing record definitions.

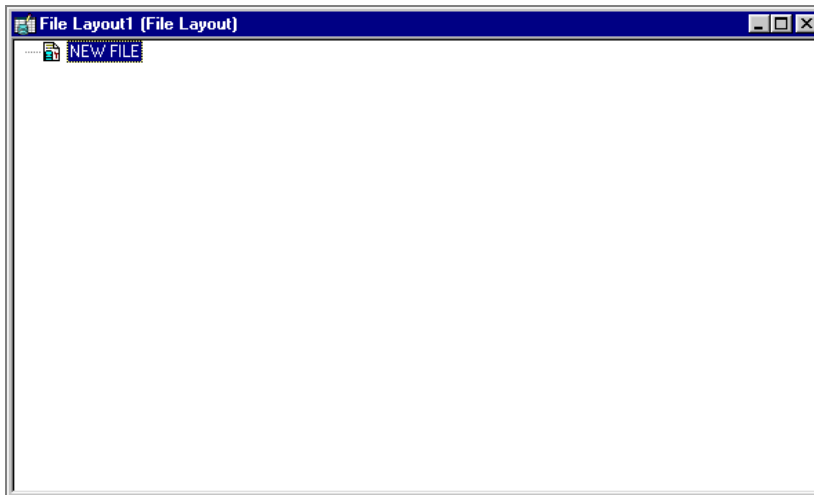
To create a new file layout

1. Select File, New, File Layout.



Creating a New File Layout Definition

Click OK to create the new File Layout.



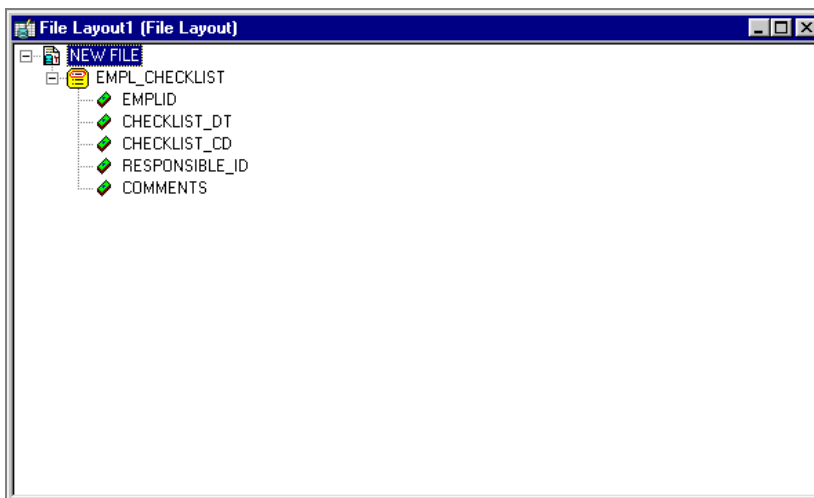
A New File Layout Definition

The default root node is “NEW FILE”. This will change to whatever you name the File Layout definition when you save the definition. Records in the File Layout can have the same name as the File Layout, however, each record must have a unique name.

The default file format type is **FIXED**.

2. Drag or Insert Records.

Records can be dragged and dropped onto the root node or inserted (using **Insert, Record**).



File Layout Definition with Single Record

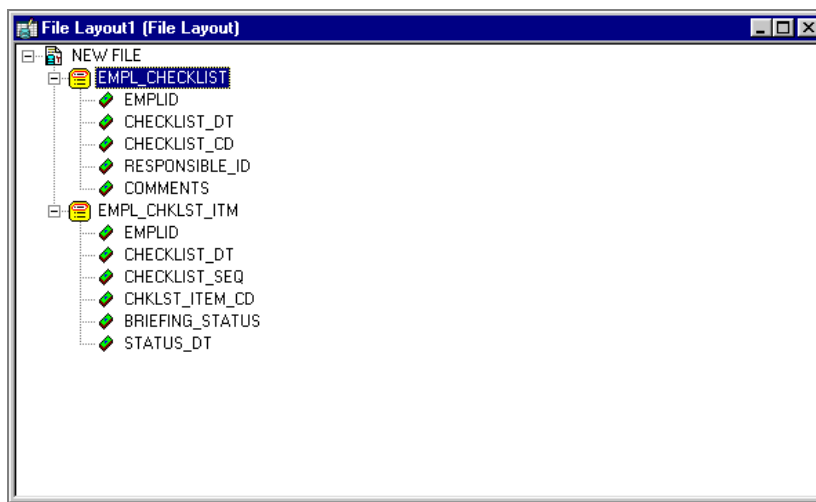
The record is used as a *template* for the File Layout. All fields are automatically expanded.



Important! A record that is dragged and dropped or inserted into a File Layout will only be used as a **template**. The record and field names will be copied to the File Record and File Field names respectively, however, **no references are stored concerning the copied object**. A change to the record will **not** be reflected in the File Layout. File Records and File Fields are stored within a File Layout only.

3. Order the records (optional).

When a second record is dragged in a File Layout, its location defaults to being a sibling of the existing File Record.



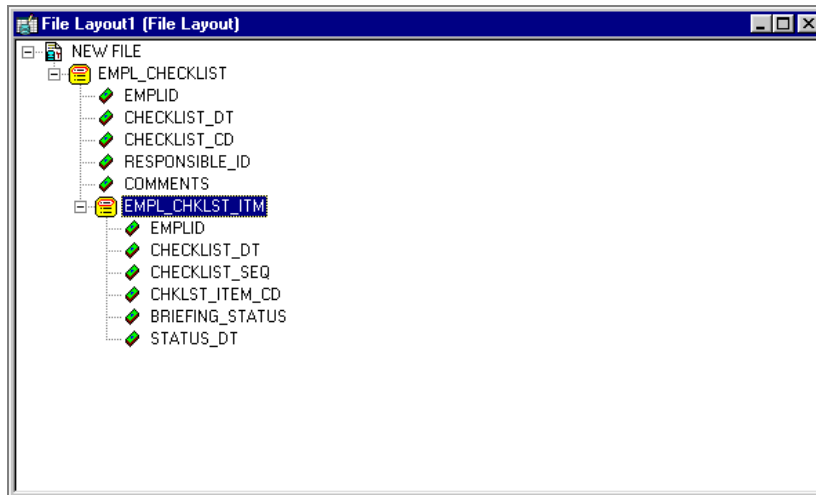
File Layout with Two Records

If you want a record to be a child of the previous File Record, select the file record you want to move, then use the order button with the right arrow on it. The order buttons are located in the Application Designer tool bar.



Order buttons

The outer buttons (up and down arrows) move the selected element's order within the File Layout. The inner two buttons (left and right arrows) change the hierarchy of the selected element. Both File Records and File Fields may be moved using the order buttons.



File Layout with Two Records in Hierarchy

Naming File Layouts, Records and Fields

File Layout names can be 30 characters in length, and should follow the PeopleSoft naming standards. File Record and File Field names can be 15 characters in length, and should follow the PeopleSoft naming standards.



For more information see Naming Records Definitions.

Each File Record within a File Layout must have a unique name. Each File Field within a single File Record must have a unique name. However, different File Records within the same File Layout can have fields with the same name.



If you use the **WriteRecord**, **ReadRowset** or **WriteRowset** file layout methods for writing to or reading from records, the application record and the File Record **must have the exact same name**. These methods only write to **like-named** records. If you rename a record after you use it to create a File Layout definition, in your File Layout definition you will have to rename your File Record to the exact same name. Because these methods use like-named records, the same file layout definition can contain more than one record. Records that aren't like-named are ignored. Like-named records do **not** have to contain all the same fields.



Note. If you use the **WriteRecord**, **ReadRowset** or **WriteRowset** file layout methods for writing to or reading from records, the application record fields and the File Fields **must have the exact same name**. These methods only write to **like-named** fields. If you rename a field after you use it to create a File Layout definition, in your File Layout definition you will have to rename your File Field to the exact same name. Fields that aren't like-named are ignored. Like-named fields do **not** have to have the same length. Peoplesoft recommends that like-named fields be of the same type.

Date, Time, and DateTime Field Considerations

When you insert fields that have a field type of Date, Time or DateTime, the field length is fixed in the File Layout.

- Date fields have a fixed length of 10
- Time fields have a fixed length of 20
- DateTime fields have a fixed length of 31

This is to follow ISO 8601 standards.



For more information about ISO 8601, see <http://www.iso.ch/markete/8601.pdf>.

When writing (outputting) a file from Peoplesoft, dates, times, and datetimes will be written in the correct, specified format.

When using a File Layout to read (input) a file, datetime fields **must** have the following format:

```
CCYY-DD-MMTHH:MM:SS.ssssss[+/-hhmm]
```

If a datetime field does **not** have this format, a NULL is written to the database, the File Layout's **IsError** property is set to True, and the field's **EditError** property is set to True.



For more information see Handling File Layout Errors.

When using a File Layout to read (input) a file, date fields **must** be in the same format as specified by the File Layout. If a date field being read doesn't have the same date format, a NULL is written to the database, the File Layout's **IsError** property is set to True and the field's **EditError** property is also set to True.



For more information see Handling File Layout Errors.

After you read data in, it's always a good practice to check the `EditError` flag to see whether there were any errors reading the data.

Considerations for Using Dates with the `ReadRowset` Method

Single digits in dates in the form `MMDDYY` or `MMDDYYYY` must be padded with zeros. That is, if the date in your data is February 3, 2000, the form must be:

`02/03/2000`

or

`02/03/00`

The following is **not** valid.

`2/3/00`

Customizing the File Layout

Each node in the File Layout (the File Layout, every File Record and every File Field) has an associated property dialog box. On these dialog boxes you can specify the format, the length and type of each node.

Some properties are only available for a specific File Layout Format. For example, a File Definition Tag is only available for a file with XML specified as the File Layout Format. When a property is only available for a particular format, that is noted in parenthesis after the name of the property (that is, File Definition Tag (XML))

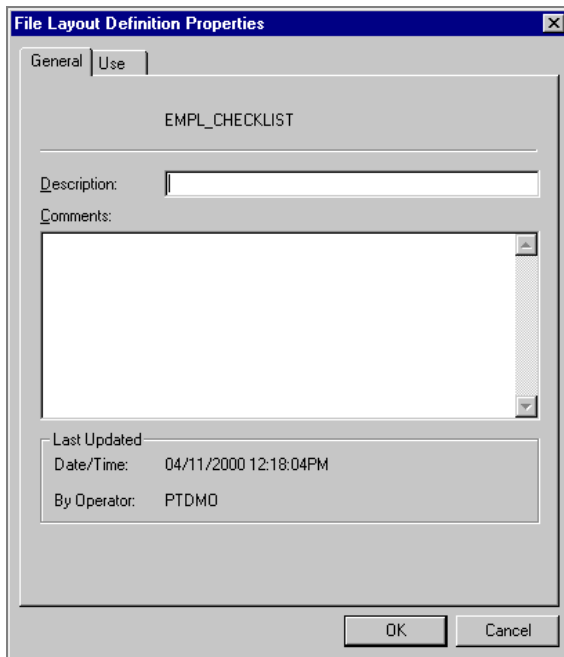
File Layout Properties

The File Layout Properties dialog box contains all information stored at the File Layout (root) level.

You can access the File Layout properties dialog box using any of the following methods:

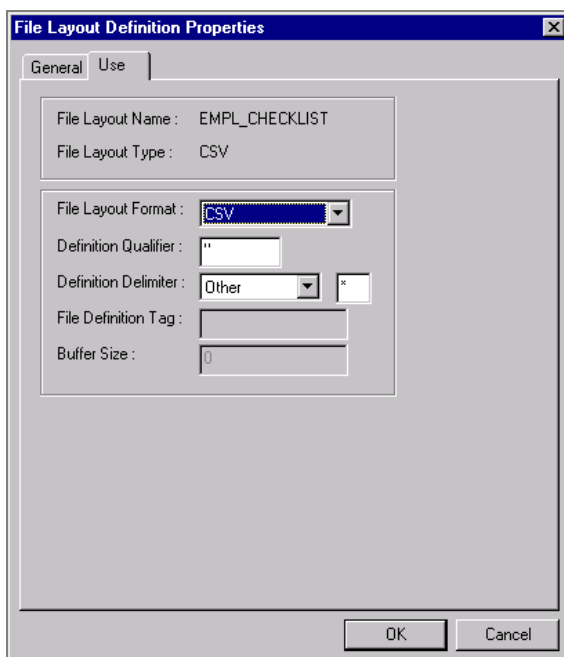
- By using File, Object Properties
- By pressing `ALT+ENTER`
- By double-clicking on the top most level of a File Layout definition
- By right-clicking on an open File Layout, then selecting **Data Object Properties**

The **General** tab of the properties dialog box contains description information for the File Layout.

The image shows the 'File Layout Definition Properties' dialog box with the 'General' tab selected. The title bar is blue with the text 'File Layout Definition Properties' and a close button. Below the title bar are two tabs: 'General' and 'Use'. The 'General' tab is active. The main area contains the text 'EMPL_CHECKLIST' in a large font. Below this is a 'Description:' label followed by a text input field. Underneath is a 'Comments:' label followed by a larger text area with a vertical scrollbar. At the bottom of the main area is a 'Last Updated' section containing 'Date/Time: 04/11/2000 12:18:04PM' and 'By Operator: PTDMO'. At the very bottom are 'OK' and 'Cancel' buttons.

File Layout Definition Properties General Tab

The **Use** tab of the properties contains specific information for the File Layout.

The image shows the 'File Layout Definition Properties' dialog box with the 'Use' tab selected. The title bar is blue with the text 'File Layout Definition Properties' and a close button. Below the title bar are two tabs: 'General' and 'Use'. The 'Use' tab is active. The main area contains several fields: 'File Layout Name : EMPL_CHECKLIST', 'File Layout Type : CSV', 'File Layout Format : CSV' (with a dropdown arrow), 'Definition Qualifier : "' (with a text input field), 'Definition Delimiter : Other' (with a dropdown arrow and a quote character in a small box), 'File Definition Tag :' (with a text input field), and 'Buffer Size :' (with a text input field). At the bottom are 'OK' and 'Cancel' buttons.

File Layout Definition Properties Use Tab

File Layout Format

The type of file layout. You can only choose one type per file layout. Valid values are FIXED, CSV, or XML.

Definition Qualifier (CSV)	The qualifier surrounding each field in the file record to use in this Layout (can be overridden at the File Record and File Field levels).
Definition Delimiter (CSV)	The default delimiter between each field in the file record to use in this Layout (can be overridden at the File Record level). The default value is Comma . If you specify Other , a blank field displays, where you can type in the delimiter you want to use.



The Definition Delimiter is overwritten by the Field Delimiter specified in the File Layout record properties.

File Definition Tag (XML)	The XML Tag name associated with this Layout (or Transaction). This tag can be 30 characters in length. This tag must be unique in the File Layout.
Buffer Size (XML)	The size of the input buffer used at runtime. You would only use this if you want to pre-allocate an input buffer. Then you should set Buffer Length equal to, or greater than, the largest record (or rowset) you expect to process.

File Record Properties

The File Record Properties dialog box contains information stored at the File Record level.

You can access the File Record properties by:

- Double-clicking on the File Record node

OR

- Selecting the File Record node, right-clicking, then selecting **Selected Node Properties**.

File Layout Record Properties

File Record Name

The File Record Name associated with this File Record. This name will be used when accessing the File Record from PeopleCode. Every File Record in a File Layout must have a unique name.



Note. If you use the **WriteRecord**, **ReadRowset** or **WriteRowset** file layout methods for writing to or reading from records, the application record and the File Record **must have the exact same name**. These methods only write to **like-named** records. If you rename a record after you use it to create a File Layout definition, in your File Layout definition you will have to rename your File Record to the exact same name. Because these methods use like-named records, the same file layout definition can contain more than one record. Records that aren't like-named are ignored. Like-named records do **not** have to contain all the same fields.

ID Seq No. (CSV)

The sequence number of the field that contains the File Record ID identifier.

Max Rec Length

The default maximum length of the combined field sizes of the record. This value isn't incremented automatically. If you add fields to a record, you will need to change this value to account for the new or changed field(s). You also need to change this value if you insert a segment then add fields to that segment.



Warning! Any inbound or outbound data will be truncated beyond this value.

File Record ID

A group of numbers used to identify the File Record. Each File Record ID must be unique in the File Layout. You can use this number in processing the file. This number is automatically written to the file if you use the **WriteRecord** or **WriteRowset** methods and the file type is FIXED or CSV.

ID Start Position

The column or starting position in the File Record where the File Record ID starts.

ID Length

The length of the File Record ID. This number is automatically generated when you enter the File Record ID. You can specify a number larger than the number of characters in a File Record ID, but you can't specify a number smaller than the number of characters in the File Record ID.

Default Qualifier (CSV)

The qualifier used for the File Record ID and the default for fields when no field qualifier is specified. This value overrides the Definition Qualifier specified in the File Layout properties dialog box. When you first create a File Layout, this property is blank.

Field Delimiter (CSV)

The delimiter used for all fields in the File Record. This overwrites the Definition Delimiter specified on the File Layout properties dialog box.

Record Tag (XML)

The XML Tag Name for this File Record. This defaults to the File Record name.



Note. Though each record name in a File Layout must be unique, record tags do **not** have to be unique.

Record Description

A description of the record. This is only for your own documentation purposes.

File Field Properties

The File Field Properties dialog box contains information stored at the File Field level.

You can access the File Field properties by:

- Double-clicking on the File Field node

OR

- Selecting the File Field node, right-clicking, then selecting **Selected Node Properties**.

File Layout Field Properties

Most of the individual properties are usable by all field types. However, some are specific to a particular field type (for example, *UpperCase* is only applicable for character fields, while *Date Separator* is only applicable for date fields, and so on.) The above screen shot is for a character type of field. However, the following description will go through all the possible properties.

Field Name The File Field Name associated with this File Field. This name will be used when accessing the File Field from PeopleCode. Every field within a File Record must have a unique name: however, two different File Records can contain the same File Field.



Note. If you use the **WriteRecord**, **ReadRowset** or **WriteRowset** file layout methods for writing to or reading from records, the application record fields and the File Fields **must have the exact same name**. These methods only write to **like-named** fields. If you rename a field after you use it to create a File Layout definition, in your File Layout definition you will have to rename your File Field to the exact same name. Fields that aren't like-named are ignored. Like-named fields do **not** have to have the same length. Peoplesoft recommends that like-named fields be of the same type.

Suppress Whether or not to suppress reading or writing this field. Primarily this is used with CSV file types, when you want to suppress reading in a select field during inbound processing. (With Fixed file types, you can specify a start position and length beyond the field and thereby "skip" it.)

Upper Case (Char) This converts lowercase text to upper case during inbound processing. This is primarily used when customer data may be in lowercase, and PeopleSoft requires the data to be in uppercase.

Field Type The data type of the File Field. The valid field types are available in a drop-down list.



For more information about data types, see Creating Field Definitions.

Date Format (Date) The format of the date, such as MMDDYY, DDYYMM, and so on. The valid date formats are available in a drop-down list.

Date Separator (Date) The character used to separate date values. Default value is /.

Decimal Position The number of decimal positions (to the right) of the decimal point. This property is only valid for fields defined as Number or Signed number.



Note. You're only allowed 31 characters for Decimal Pos (31 characters plus a decimal point.)

Field Length

The length (maximum number of characters) of this field.



Note. You're only allowed 32 character precision for number and signed number fields, that is, a **total** of 32 characters both to the right and left of the decimal. Other fields, such as character fields, can have a longer length.



Note. You will **not** be able to set the field length for fields of type Date, Time, and DateTime. These field lengths are automatically set to the ISO standards for such fields.

Start Position (FIXED)

The starting position (column) of the field within the File Record.



Important! If you specify a Start Position for a field that overwrites a previous field, **no data** will be written to the file. Use **Propagate** to change the Start Positions for your File Fields.

Propagate (FIXED)

If a field position or length is changed, an amount may be entered here to increment (positive number) or decrement (negative number) the current field and all fields before it (<<<) or after it (>>>).

Field Qualifier (CSV)

The qualifier for the field, that is, the character that surrounds this field, separating it from other fields. Specifying this value will overwrite the value specified in the File Layout properties **and** File Record properties.

Field Tag (XML)

The XML Tag Name to be used around the field. The default value is the name of the field.



Note. Though each field name in a File Record must be unique, each Field tag does **not** have to be unique.

Strip Chars

Specify any characters to be removed from the input buffer. You would use this to preprocess input strings. For example, if this field in your input file contains hyphens, but you want to remove the hyphens prior to doing any processing on the field, you could enter a hyphen here, and it would be stripped out while being read. You can specify more than one character to be stripped out. Be sure to not separate the strip characters. For example, the following will strip out all hyphens and semi-colons:

-;

The following will strip out all hyphens, semi-colons **and** spaces:

; -

Field Description

A description of the field. This is for your own documentation purposes.

Field Inheritance

Inheritance allows the field’s value to be derived from the specified Parent File Record and Field. If you’re writing to a file, specifying a record and field name means the value will only be written in the parent File Record, not the child (inheriting) File Record (that is, the value won’t be written more than once to a file.)

For example, the following file sample shows both the EMPLID (8113) and EFFDT (08/06/1999) written only once to a file, though these fields are repeated in the third File Record (with File Record ID 102.)

```
100 8113      Frumman,Wolfgang
101      08/06/1999 000001      8219  Going to London office
102      100      000015 I 08/06/1999
102      200      000030 I 08/06/1999
102      300      000009 I 08/06/1999
102      400      000001 I 08/06/1999
102      500      000011 I 08/06/1999
```

Record Name

Specify the name of the record you want to inherit a value from.

Field Name

Specify the name of the field on the record that you want to inherit a value from.

Default Value

If no value is present, the value specified here will be used.

Using Segments instead of Records

Instead of inserting records, you can insert segments and fields into a record. What is the difference between a record and a segment? When you insert a record into a File Layout, all the fields are inserted, along with additional information. A segment is the exact same as a record, only it contains no information. You have to add all the information yourself. After you add the information to a segment, you can treat it just like any other file record.

For example: suppose in the file provided to you, some of the File Records contain new data, and need to be inserted, while others contain data that's updating existing data. You could add a segment with a single field that indicates whether the Field Record was new or changed (like `AUDIT_ACTION`). When you process the file, you can use PeopleCode to look at this field, and based on its value, do the appropriate action.

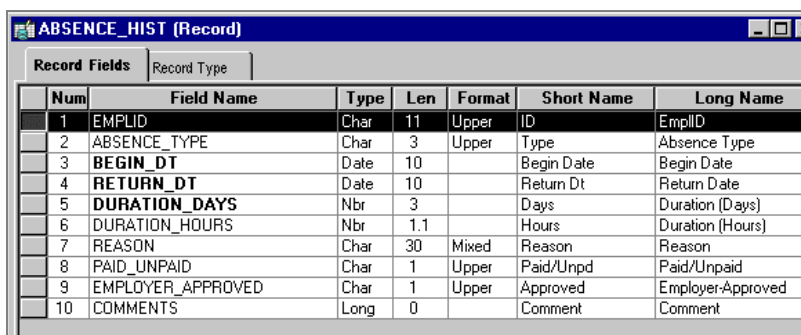
Another example: suppose you wanted to include two fields from the `PERSONAL_DATA` table in your file, but not all the other fields. You have two choices: insert the `PERSONAL_DATA` table and manually delete all the unwanted fields, or insert a segment, name it `PERSONAL_DATA`, then insert the two fields you want.

File Layout Example

The following example will illustrate creating a File Layout that could be used with the `ABSENCE_HIST` record. In addition, this example will insert a segment used to track `AUDIT_ACTION`, with the following meanings:

- A - Row inserted
- C - Row changed (updated), but no key fields changed.

To create a File Layout definition



Num	Field Name	Type	Len	Format	Short Name	Long Name
1	EMPLID	Char	11	Upper	ID	EmpID
2	ABSENCE_TYPE	Char	3	Upper	Type	Absence Type
3	BEGIN_DT	Date	10		Begin Date	Begin Date
4	RETURN_DT	Date	10		Return Dt	Return Date
5	DURATION_DAYS	Nbr	3		Days	Duration (Days)
6	DURATION_HOURS	Nbr	1.1		Hours	Duration (Hours)
7	REASON	Char	30	Mixed	Reason	Reason
8	PAID_UNPAID	Char	1	Upper	Paid/Unpd	Paid/Unpaid
9	EMPLOYER_APPROVED	Char	1	Upper	Approved	Employer-Approved
10	COMMENTS	Long	0		Comment	Comment

ABSENCE_HIST record definition

For simplicity, let's say each row in our data file has the following structure:

```

888 A
000 8001 VAC 1981-09-12 1981-09-26 14 .0 P Y
888 A
000 8001 VAC 1983-03-02 1983-03-07 5 .0 P Y
888 A
000 8001 VAC 1983-08-26 1983-09-10 13 .0 P Y
888 A
000 8516 MAT 1986-06-06 1986-08-01 56 .0 P Y
888 C
000 8516 SCK 1988-08-06 1988-08-07 1 .0 P Y
888 A
000 8516 VAC 1987-07-14 1987-07-28 14 .0 P Y
888 A
000 8553 JUR 1990-12-12 1990-12-17 5 .0 Local Jury Duty P N
888 A
000 8553 MAT 1992-02-20 1992-10-01 224 .0 Maternity Leave U N
888 A
000 8553 MAT 1994-08-19 1995-03-01 194 .0 Maternity U Y
888 A
000 8553 PER 1993-04-15 1993-04-19 4 .0 U N Personal Day
required
888 C
000 8553 SCK 1987-01-28 1987-01-30 2 .0 Hong Kong Flu P N
888 A
000 8553 SCK 1988-08-02 1988-08-03 1 .0 Sick P N
888 A
000 8553 SCK 1995-09-12 1995-09-13 1 .0 P N
888 C

```

```
000 G001 MAT 1991-07-02 1991-09-28 88 .0 3-month Maternity P Y Maternity
will be paid as 80% of Claudia's current salary.
```

000 is the File Record ID for ABSENCE_HIST and each field appears in the same order as in the ABSENCE_HIST record. **888** is the File Record ID for the CHANGE_ACTION segment.

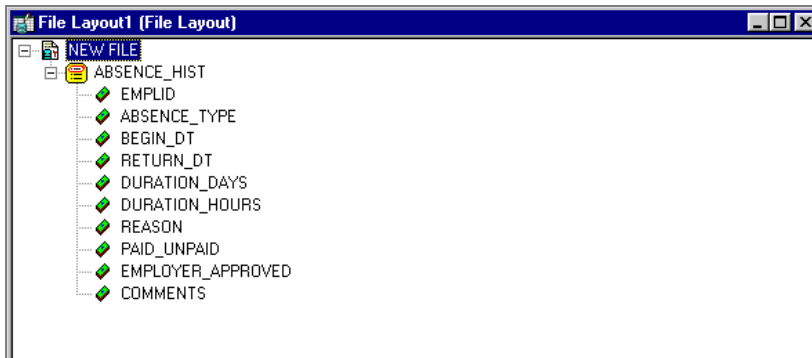


The end of file (EOF) character **must** be on a separate line and not on a line containing data for any incoming file, regardless of file type. Each data line needs to be terminated with an end of line (EOL) character, which is different than an EOF.

This file is of type FIXED.

1. Use the ABSENCE_HIST record definition as a template for the File Layout.

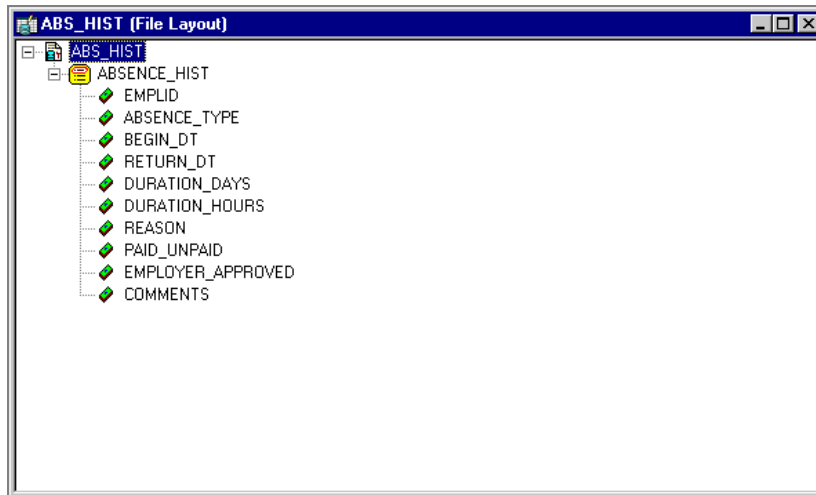
Create a new File Layout, then drag the ABSENCE_HIST record into the open File Layout.



File Layout with ABSENCE_HIST record

2. Save the File Layout.

We'll save it with a name of ABS_HIST.



File Layout saved as ABS_HIST

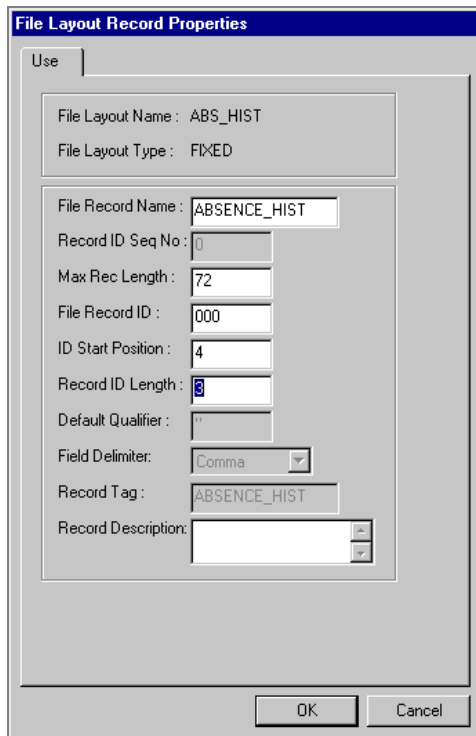
Notice that the first node has changed from NEW FILE to ABS_HIST.

3. Change the File layout properties.

Double-click on the top-most node in the File Layout, ABS_HIST, to display the File Layout properties. Fill in a short and long description of the file layout you're creating. For this example, we're creating a FIXED file layout, so you don't need to make any changes on the Use tab.

4. Change the File Record properties.

Double-click on the ABSENCE_HIST File Record to display the File Record Properties. Enter a Record ID of **000**, a Starting Position of 1 and a Length of 4:



The dialog box is titled "File Layout Record Properties". It has a "Use" tab selected. Inside, there are several fields and a dropdown menu:

- File Layout Name: ABS_HIST
- File Layout Type: FIXED
- File Record Name: ABSENCE_HIST
- Record ID Seq No: 0
- Max Rec Length: 72
- File Record ID: 000
- ID Start Position: 4
- Record ID Length: 8
- Default Qualifier: "
- Field Delimiter: Comma (dropdown menu)
- Record Tag: ABSENCE_HIST
- Record Description: (empty text box)

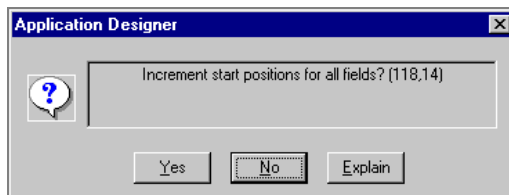
At the bottom are "OK" and "Cancel" buttons.

File Layout Record Properties Example



Note. If you forget to update the Record ID Length field you'll receive an error message.

When you click OK, you'll get the following message:



The dialog box is titled "Application Designer". It contains a question mark icon and a text box with the message: "Increment start positions for all fields? (118,14)". At the bottom are three buttons: "Yes", "No", and "Explain".

Increment Field Position message

Click Yes. This will automatically increment the start position numbers for every field to take the length of the file Record ID you just added into account. If you don't click yes, you will have to manually increment the start position for all your fields.

We have just created the File Record ID for the ABSENCE_HIST record (**bold** text from our sample file above):

```

000 8001 VAC  1981-09-12 1981-09-26 14  .0          P Y
000 8001 VAC  1983-03-02 1983-03-07 5   .0          P Y

```




Note. If you use the **WriteRecord**, **ReadRowset** or **WriteRowset** file layout methods for writing to or reading from records, the application record and the File Record **must have the exact same name**. These methods only write to **like-named** records. If you rename a record after you use it to create a File Layout definition, in your File Layout definition you will have to rename your File Record to the exact same name. Because these methods use like-named records, the same file layout definition can contain more than one record. Records that aren't like-named are ignored. Like-named records do **not** have to contain all the same fields.

5. Change the File Field Properties.

When a record definition is used as a template for a File Layout, each field's starting position is defaulted based on the order it appears in the record as well as its length.

Double-click on the EMPLID File Field to display its Properties:

File Layout Field Properties Example

Notice that the Start Position is automatically incremented to 5 (since the File Record ID is 4 character long.) However, in the example there's an extra space between the end of the File Record ID and the first field. Therefore, you need to change the start position of this field, and all the fields after this field. To do this:

- Click once on the up arrow under Propagate, to change that number from 0 to 1

- Click the button with the arrows pointing right (>>>).

This will increment the starting position of this field and all fields following this field by 1.

6. Adjust other fields (optional)

If you look carefully at the record definition for ABSENCE_HIST, you'll notice that the last field (COMMENTS) has a length of 0. This is because it's a field of type Long.



All Long fields have a default length of 0.

Because the format for this File Layout is FIXED, you have to change the Field Length of that field to something other than 0 if you want to pick up the data for that field.

File Layout Field Properties

Use

Field Inheritance
File Layout Name : ABS_HIST
File Layout Type : FIXED

Sequence No : 10

Field Name : COMMENTS ☐ Suppress

Field Type : Character ☐ UpperCase

Decimal Pos : 0

Field Length : 0

Start Position : 80

Field Qualifier : *

Field Tag : COMMENTS

Strip Characters : ☐ Trim Spaces

Field Description :

Field Inheritance
Record Name : DO-NOT-INHERIT
Field Name : COMMENTS
Default Value :

Propagate: <<< 0 >>>

OK Cancel

Example File Layout Field Properties

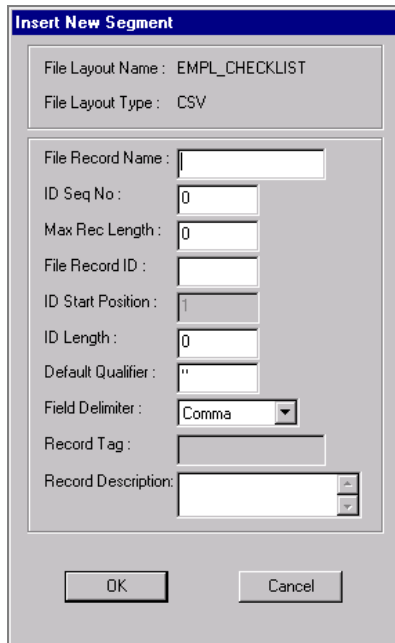
You don't have to propagate this change because this is the last field in the record.

7. Increase the Max Rec Length for the File Record

The **Max Rec Length** field on the File Record properties sheet does **not** automatically get updated to reflect any changes that you may make to a field length. After you change any field lengths, you must go back and update the **Max Rec Length** on the File Record property sheet to the appropriate value.

8. Insert a segment.

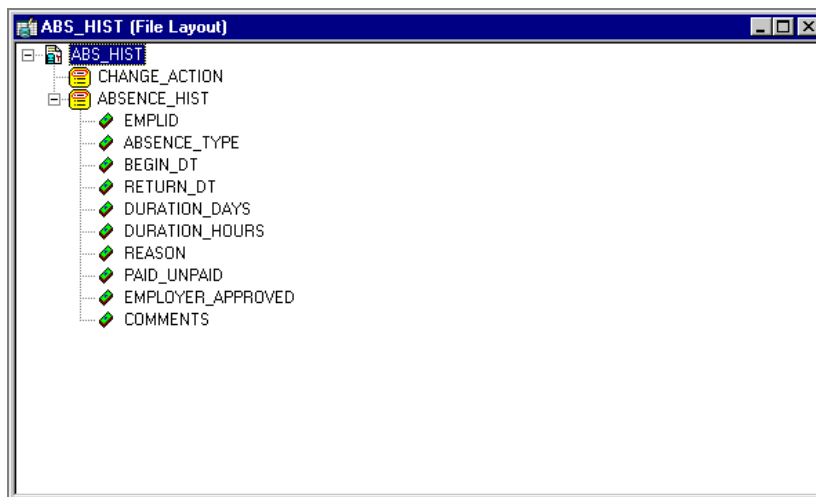
We want to insert a segment that is a sibling, (that is, at the same level), as the ABSENCE_HIST record. Insert a segment by selecting **Insert Segment** from the pop-up menu, or by going **Insert, Segment**. The Insert New Segment dialog box is displayed.



The "Insert New Segment" dialog box is shown. It has a title bar "Insert New Segment". Inside, there are two sections. The top section contains "File Layout Name : EMPL_CHECKLIST" and "File Layout Type : CSV". The bottom section contains several input fields: "File Record Name : " (empty), "ID Seq No : 0", "Max Rec Length : 0", "File Record ID : " (empty), "ID Start Position : 1", "ID Length : 0", "Default Qualifier : " (empty), "Field Delimiter : Comma" (dropdown menu), "Record Tag : " (empty), and "Record Description : " (empty). At the bottom are "OK" and "Cancel" buttons.

Inserting New Segment dialog box

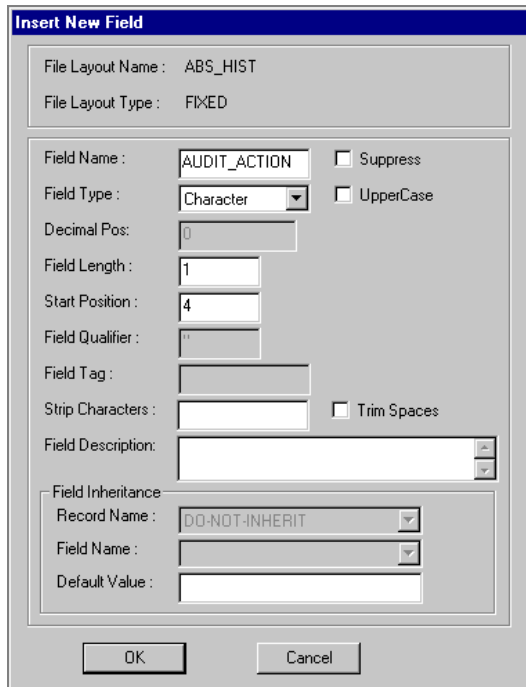
Does this dialog box look familiar? It should. It's identical to the File Layout Record properties dialog box. Fill in the **File Record Name**, **Max Rec Length**, **File Record ID**, **ID Start Position** and **Record ID Length** attributes. When you click OK, the segment will be inserted.



File Layout with Inserted Segment

9. Insert a field.

Now we want to insert a File Field into the segment. Insert a field by selecting **Insert Field** from the pop-up menu, or by going **Insert, Field**. The Insert New Field dialog box is displayed.

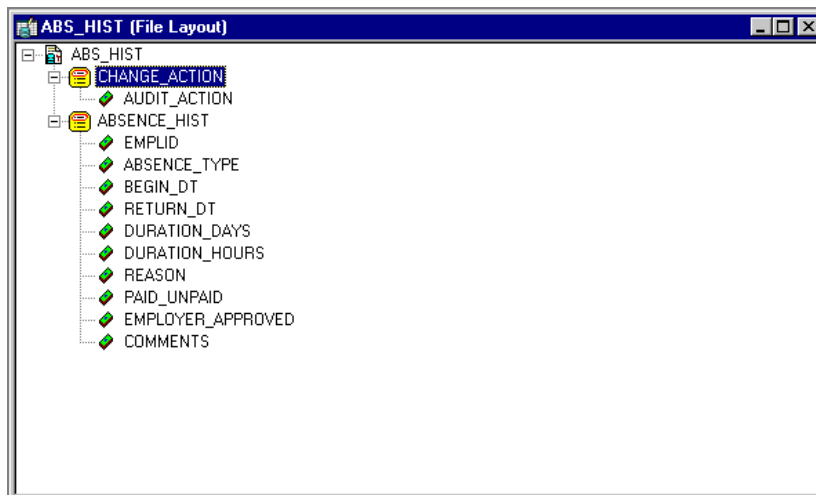


The **Insert New Field** dialog box is shown. It contains the following fields and options:

- File Layout Name :** ABS_HIST
- File Layout Type :** FIXED
- Field Name :** AUDIT_ACTION
- Field Type :** Character
- Decimal Pos:** 0
- Field Length :** 1
- Start Position :** 4
- Field Qualifier :** ''
- Field Tag :**
- Strip Characters :**
- Field Description:**
- Field Inheritance:**
 - Record Name :** DO-NOT-INHERIT
 - Field Name :**
 - Default Value :**
- Options:**
 - ☐ Suppress
 - ☐ UpperCase
 - ☐ Trim Spaces
- Buttons:** OK, Cancel

Insert New Field dialog box

Again, a familiar dialog box! Fill in the **Field Name**, **Field Length**, and **Start Position** of the new field, then click OK.



File Layout with new field added

10. Save your work.

Be sure to save the changes you've made to your File Layout by going to File, Save, or clicking on the save icon in the toolbar.

Now that you've created and saved a File Layout, you must use PeopleCode to access the data. File Layouts rely solely on PeopleCode as the engine behind the actual data access and movement. This powerful interface allows application developers to access data from a file as they would a message or a panel buffer (scroll). Using methods such as **WriteRecord** or **ReadRowset** there is no need to parse each File Record into fields.



For more information see File Class.

Supported File Formats

The following formats are supported by File Layout definitions.

- Fixed Format Positional File (FIXED) (this is the default)
- Variable Format Delimited File (CSV)
- Tagged Hierarchical Data File (XML)

Each File Layout only has one format type associated with it.

Fixed Format Positional File

This is the most common type of flat file currently processed by EDI Manager. Almost all EDI type processing (as it exists today) utilizes this file type where each data element is oriented by a fixed, or column dependent, position within the file.

Attribute	Description	EDI Manager Equivalent
File Layout Format	The File Format (FIXED).	none
File Record ID	A group of numbers that can be used to identify the File Record.	RowID
ID Start Position	The column starting position of the File Record ID.	Treated as a Field within each Map
(Record) ID Length	The length of the File Record ID.	Treated as a Field within each Map
File	A user specified name for the File Record.	PeopleSoft Record

Attribute	Description	EDI Manager Equivalent
Record Name		Name
File Field	A user specified name for the File Field.	PeopleSoft Record's Field Name
(Field) Start Position	The column starting position of the File Field.	Starting Position
Field Length	The length of the File Field.	Length of Field
Field Format	The formatting options for both inbound and outbound field processing.	Based on Field type

Considerations using FIXED Format

You should be aware of the following when working with files of FIXED format.

- Be careful when you change the length or starting position of any file fields, or if you insert a new file field between two existing ones. It's possible to overlay fields. Results are unpredictable.
- When you insert a record into a File Layout, be sure to check the length of any fields of type Long. It's legal for these fields to have a length of 0. You will need to change that length to the appropriate length (256, generally) if you want that information included in your file.
- Max Record Length is calculated only for file records, and only once, when the file record is first added to the File Layout. It isn't dynamically calculated after that. If you add or delete any file fields, or change the length of any file fields, you will also have to change the Max Record Length.
- Max Record Length is never calculated for segments. You will always have to total the length of any file fields for a segment.

Variable Format Delimited File (CSV)

In this type of file, each data element is surrounded with a separator, a delimiter or both. File Record IDs can be used to determine which table data is moved to or from, however, in most cases this type of file contains homogenous records.

If your text contained the following, the qualifier would be double quotes (") and the delimiter would be a comma (,).

```
"NAME" , "ADDRESS" , "PHONE"
```

File Layout definitions store the File Record Identifier (when used) and the relative sequence number of the field. (In the text example above, "PHONE" would be sequence number 3).

Attribute	Description	EDI Manager CSV Equivalent
File Format	The File Format (CSV).	none
File Record ID	A group of numbers that can be used to identify the File Record.	RowID
ID Sequence Number	The sequence number of the Field that contains the Record identifier.	Treated as a Field within each Map
Field Qualifier	The qualifier used when processing this file. This can be set at the File Layout, File Record or File Field level.	Delimiter
Field Delimiter	The delimiter used when processing this file. This can be set at the File Layout or File Record level.	Separator
File Record Name	A user specified name for the File Record.	None
File Record Field	A user specified name for the File Field.	None
Field Format	The formatting options for both inbound and outbound field processing.	Based on Field type

Considerations using CSV Format

You should be aware of the following when working with files of CSV format.

- Both the Definition Qualifier and the Definition Separator accept a blank as a valid value.
- If a field is NULL, you don't have to use qualifiers. In the following example, Field2 is NULL.

```
Field1,,Field3,Field4. . .
```

Tagged Hierarchical Data File (XML)

This type of file contains data represented in a hierarchical or 'tree' type structure. A tag surrounds each data element. A File Record tag might group multiple entries.

File Layout definitions tie the identifier along with parent and child relationships to the File Record and File Field.

There is no EDI Manager Equivalent for this format.

Attribute	Description
File Format	The File Format (XML).
File Record ID	The Tag Name representing the File Record.
Field Identifier	The Tag Name representing the File Field.

Attribute	Description
Field Format	The formatting options for both inbound and outbound field processing.
File Record Name	A user specified name for the File Record.
Field Name	A user specified name for the File Field.

Considerations using XML Format

You should be aware of the following when working with files of XML format.

- If you use the WriteRecord method to write data to your file, you will have to add your own end tag to indicate the end of the record. This is because the code has no way of knowing whether you want to write children records following the record just written out.

In the following example, the text in **bold** has to be written to the file using the WriteLine method because it isn't automatically added by WriteRecord:

```
<RECORD1>
```

```
Text
```

```
<RECORD2>
```

```
More text
```

```
</RECORD2>
```

```
</RECORD1>
```

- In most cases, you should use the Buffer Length that's automatically generated and used internally. However, if you want to pre-allocate an input buffer, you should set Buffer Length equal to, or greater than, the largest record (or rowset) you expect to process.
- Max Record Length is calculated only for file records, and only once, when the file record is first added to the File Layout. It isn't dynamically calculated after that. If you add or delete any file fields, or change the length of any file fields, you will also have to change the Max Record Length.
- Max Record Length is never calculated for segments. You will always have to total the length of any file fields for a segment.

CHAPTER 5

Open Query ODBC Driver and API

This section describes the PeopleTools Open Query Interface, an API based on the defacto data access standard ODBC.

Overview

Features

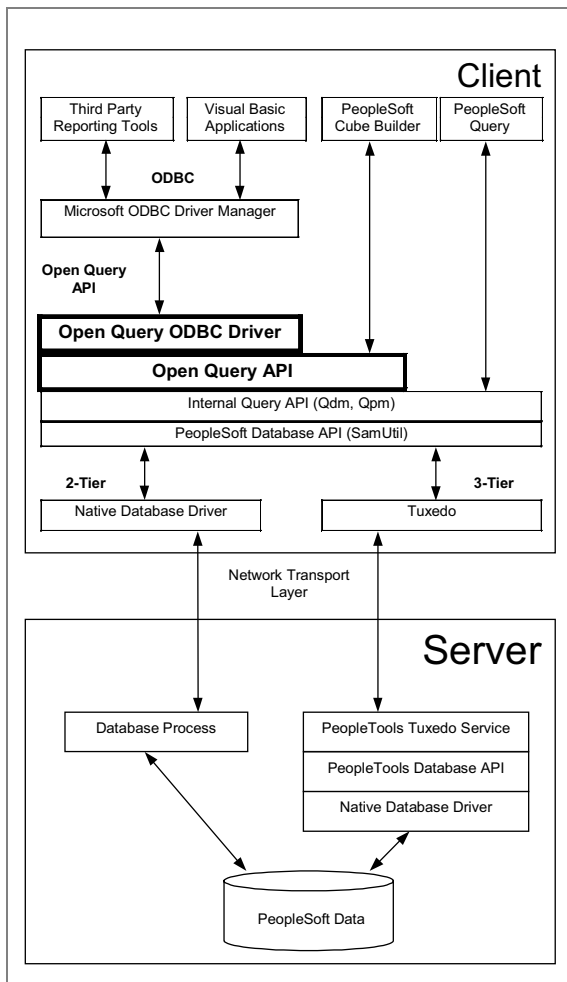
The PeopleSoft Open Query ODBC driver and API have been designed to allow third party reporting tools or applications to access PeopleSoft data in conformance with the PeopleSoft Query Access Architecture (the embedded SQL access intelligence provided by PeopleSoft Query). The Query Access Architecture provides the following key features:

- Multiple levels of security
 - Query authorization
 - Operator security
 - Operator profile
 - Record Level
 - Access group
 - Row Level
 - Security record
- Standard Query data access
 - Access to all supported PeopleSoft databases
 - Ability to run stored PeopleSoft queries
 - Automatic use of TableSets
 - Effective dated output
 - Translate values

- Effectively dated
- International translations

Architecture

The following diagram illustrates the components involved in the Open Query Architecture. The bold blocks represent the components of the Open Query product and are the focus of this document.



Open Query Architecture

The two bolded blocks above are the deliverables described below.

Product	Audience	Purpose
Open Query ODBC	Third party application vendors and implementation partners	Open, documented API providing access to PeopleSoft Query as a data source. Primary means of achieving

		programmatic interface to PeopleSoft data.
Open Query API	PeopleTools Development	PeopleSoft proprietary. Exposes Query definition and run-time functions for use <u>within</u> PeopleTools. API used by ODBC driver and OLAP.

Components

Open Query ODBC Driver

The Open Query ODBC driver represents the external means to extract data from PeopleSoft database.

Using ODBC as our external API has several distinct advantages:

- Third party tools do not need special information about PeopleSoft data
- Most third party reporting and query tools already support ODBC
- Custom integration with PeopleTools no longer required.
- Eliminates proprietary interface drivers (namely p2sps.dll)
- Supported by other application development tools, like Visual Basic and PowerBuilder
- Connectivity to PeopleSoft data is maintained by the PeopleSoft security architecture
- Extension or modification of driver behavior is allowed via the ODBC standard

The ODBC driver does not have any intrinsic knowledge of PeopleSoft data structures. Only the interface components exposed via the Open Query API are required to extract ad-hoc query data.

This layer provides only the data modification and conversion code necessary to comply with the ODBC SDK standards. None of the PeopleTools structures are exposed at this level.

Open Query API

Whereas the ODBC driver is designed to present a stable interface to external applications, the Open Query API is constantly under construction and therefore is not useful in this capacity. From earlier experience, the requirement to modify behavior between releases presents a challenge when dealing with third party applications. The solution was to create an intermediate layer between the ODBC driver and the underlying PeopleTools manager code. This wrapper code reduces the number of method calls to an acceptable level and allows the PeopleTools development team some leeway when new functionality is required or when the underlying code base is modified.

The Open Query API allows the external driver (ODBC) and PeopleTools applications to focus solely on providing PeopleSoft data in the formats described by those products. It also abstracts the underlying connective architecture from the upper levels of the interface.

As the architecture of this API is fluid, it is only available to PeopleTools development.

External Reporting Tools

The Open Query ODBC driver will allow third party reporting tools or ODBC-enabled development applications to access a PeopleSoft database. The driver will enforce user, table, and row level security through internal PeopleTools APIs. A user can leverage PeopleSoft Query to easily create platform-independent queries. These ad-hoc queries can be run against any supported PeopleSoft database platform.

Perhaps the greatest benefit of the ODBC approach is the ability to choose the best of breed reporting or analysis tool. Users will be able to use their tool of choice.

Internet

The Open Query API is a valuable link to all external data access mechanisms, including access to data over the internet.

Supported ODBC v2.5 Functions

This quick-reference summary lists the ODBC API calls supported by the ODBC driver. API calls not supported will return SQL_ERROR. Each call is discussed in further detail in the following section.

<i>ODBC Call</i>	<i>Function</i>
SQLAllocEnv	Allocate an environment handle for the ODBC connection.
SQLAllocConnect	Allocate a connection; returns a connection handle.
SQLAllocStmt	Allocate a statement handle for the specified connection.
SQLBindCol	Provide the buffer address for an answer column about to be fetched.
SQLBindParameter	Provide the value of a parameter (prompt variable) defined in the Query.
SQLColAttributes	Returns result column descriptor information for a result set.
SQLConnect	Connect to the PeopleSoft database.
SQLDescribeCol	Provide descriptors (data type, etc.) for a result column.
SQLDescribeParam	Describe a parameter marker in a statement.
SQLDisconnect	Disconnect from the data source.
SQLDriverConnect	Connect to the PeopleSoft database, prompting the user for

	any logon parameters not provided by the caller.
SQLError	Retrieve information about an error that occurred on a previous call.
SQLExecDirect	Prepare and execute a Query. Note: only the stored procedure syntax is supported in phase 1.
SQLExecute	Execute a previously prepared Query.
SQLFetch	Fetch a row of the answer set into the bound columns.
SQLFreeConnect	Close the database connection and free all resources associated with it.
SQLFreeStmt	Discard all resources associated with a previously prepared statement.
SQLGetData	Retrieve data for a specific column of the current fetched row. (Useful for long data, images, etc.)
SQLGetFunctions	Tell applications what ODBC functions this driver supports.
SQLGetInfo	Retrieve information about the data source.
SQLGetRowCount	Return the number of rows affected by the last execution.
SQLGetTypeInfo	Return information about data types supported by the data source.
SQLNumParams	Return the number of parameters in a statement.
SQLNumResultCols	Return the number of result columns in the answer set of a prepared Query.
SQLPrepare	Prepare a Query for execution.
SQLProcedureColumns	Provide a list of Queries and result columns available to the current operator and matching the specified qualifiers.
SQLProcedures	Retrieve a list of available stored procedures (Queries).
SQLTransact	Commit or rollback the current transaction.

The following table of ODBC functions are supported calls with no underlying functionality. These functions exist to ensure compatibility with ODBC applications.

ODBC Call	Function
SQLColumns	Retrieve column information from the database.
SQLForeignKeys	Retrieve database information concerning foreign keys.
SQLGetConnectOption	Get connection option information.
SQLGetCursorName	Get the name of the cursor.
SQLGetStmtOption	Get statement option information.
SQLMoreResults	Returns whether or not another result set is pending.

SQLPrimaryKeys	Retrieve database information on primary keys.
SQLSetConnectOption	Set database connection options.
SQLSetCursorName	Set the name of the cursor to be used with the statement.
SQLSetScrollOptions	Set options to control cursor scrolling.
SQLSetStmtOption	Set options for the statement.
SQLSpecialColumns	Retrieve information about optimal keys or auto incremented columns.
SQLStatistics	Retrieve statistics on tables and indices from the database.
SQLTables	Retrieve a list of tables and/or views in the database.

ODBC Driver Application Flow

This section provides a detailed explanation of how the ODBC data access model maps to the PeopleSoft data access model. For an introduction to ODBC standards and levels of compliance, see Appendix A. The areas of ODBC functionality are divided into five categories:

- Connectivity
- Execution
- Data Retrieval and Cursor Model
- Data Types
- Conformance Level

For each category, we give an overview of the ODBC model and supporting API functions, then explain how the PeopleSoft Static Query ODBC driver will implement the ODBC functions. If you are already familiar with ODBC, skip to the **PeopleSoft Driver** section. This section is specific to the PSQODB32 driver.

The Static Query ODBC driver will implement most, but not all, functions of a Level 1 API compliant ODBC driver. The driver will only support a pre-defined PeopleSoft Query object. A PeopleSoft Query object will be exposed to the application developer through the ODBC stored procedure API.

Initializing PeopleTools

SQLAllocEnv

SQLAllocEnv allocates memory for an environment handle and initializes the ODBC call level interface for use by an application.

```
RETCODE SQLAllocEnv (phenv)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HENV FAR *	Output	Pointer to storage for the environment handle.

SQLAllocConnect

SQLAllocConnect allocates memory for a connection handle within the environment identified by henv. This is called after SQLAllocEnv.

RETCODE **SQLAllocConnect** (henv, phdbc)

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HENV	Input	Environment handle.
2	HDBC FAR *	Output	Pointer to storage for the connection handle.

SQLFreeEnv

SQLFreeEnv releases an environment handle and frees all memory associated with the handle. This is called after SQLFreeConnect.

RETCODE **SQLFreeEnv** (henv)

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HENV	Input	Environment handle.

Connection Model

ODBC uses an abstraction that maps a single name (called the data source name or DSN) to all the necessary underlying software components required to access the data. The data source name is chosen by an end user or a system administrator and should be a name that makes clear to the user what kind of data it represents. ODBC data source mapping information is maintained in the registry in Windows NT and Windows 95.

In order to connect to a PeopleSoft data source, several pieces of information are needed. With the introduction in PeopleTools 7 of multiple signon capabilities, it is necessary to prompt the user for the required logon information. By default, PeopleTools installs an ODBC data source with the name **PeopleSoft PeopleTools**. This DSN has no references to PeopleSoft connection information. It is in effect an empty data source. Using this data source forces the underlying PeopleSoft security mechanisms to present the user with a signon dialog. The user fills this in as they would for a normal PeopleSoft application.

As per the ODBC standard, the PeopleSoft driver allows the user to create a data source that provides the information necessary to complete a logon. If the information matches a current logon session, the user will not be prompted to logon again.

It is worthwhile to note that the connection environment is affected by the workstation settings for the Process Scheduler. The values for the DBBIN and TOOLBIN are searched for the necessary support files required to logon to a database. Hence these values need to be valid. In the case of a 3-tier logon, the value for DBBIN can be and should be set empty.

ODBC API Functions

ODBC supports three connection functions: *SQLConnect*, *SQLDriverConnect*, and *SQLBrowseConnect*. *SQLBrowseConnect* will not be supported by the query ODBC driver.

SQLConnect

RETCODE **SQLConnect**(hdbc, szDSN, cbDSN, szUID, cbUID, szAuthStr,cbAuthStr)

Argument	Type	Use	Description
1	HDBC	Input	Connection handle.
2	UCHAR FAR*	Input	Data source name.
3	SWORD	Input	Length of szDSN
4	UCHAR FAR*	Input	User identifier.
5	SWORD	Input	Length of szUID
6	UCHAR FAR*	Input	Authentication string (typically the password).
7	SWORD	Input	Length of szAuthStr.

SQLConnect loads a driver and establishes a connection to a data source. The connection handle references storage of all information about the connection, including status, transaction state, and error information.

This function assumes a connection may be completed by supplying only a DSN, user, and password. It is further assumed that the application will either prompt the end user for security information, the security information is hard-coded or that the security information can be obtained from a centralized security server like Kerberos.

SQLDriverConnect

RETCODE **SQLDriverConnect**(hdbc, hwnd, szConnStrIn, cbConnStrIn, szConnStrOut, cbConnStrOutMax, pcbConnStrOut, fDriverCompletion)

Argument	Type	Use	Description
1	HDBC	Input	Connection handle
2	HWND	Input	Window handle. The application can pass the

			handle of the parent window, if applicable, or a null pointer if either the window handle is not applicable or if <code>SQLDriverConnect</code> will not present any dialog boxes.
3	UCHAR FAR*	Input	A full connection string, a partial connection string, or an empty string.
4	SWORD	Input	Length of <code>szConnStrIn</code> .
5	UCHAR FAR*	Output	Pointer to storage for the completed connection string. Upon successful connection to the target data source, this buffer contains the completed connection string. Applications should allocate at least 255 bytes for this buffer.
6	SWORD	Input	Maximum length of the <code>szConnStrOut</code> buffer.
7	SWORD FAR*	Output	Pointer to the total number of bytes returned in <code>szConnStrOut</code> . If the number of bytes is \geq <code>cbConnStrOutMax</code> , the completed connection string in <code>szConnStrOut</code> is truncated to <code>cbConnStrOutMax - 1</code> .
8	UWORD	Input	Flag which indicates whether Driver Manager or driver must prompt for more connection information: SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED, or SQL_DRIVER_NOPROMPT.

SQLDriverConnect handles the entire connection process for an application, prompting the end user for any necessary information to complete connection. The programmer can specify as much or as little about the connection as he or she wants to. In the simplest case, the application doesn't specify any information at all about the connection. It simply supplies the connection handle returned from *SQLAllocConnect*, a window handle, and option specification of *SQL_DRIVER_COMPLETE*, and zeros or NULLs for the rest of the arguments:

```
rc = SQLDriverConnect(hdbc, hwnd, NULL, 0, NULL, 0, 0, SQL_DRIVER_COMPLETE);
```

PeopleSoft Driver

The following keywords are used and supported by the PeopleSoft ODBC driver:

- **DSN** – Data Source Name required by ODBC
- **APPNAME** – Application Server name used for 3-tier logon only
- **DBTYPE** – Database type of logon can be any of the following values

- DB2 – DB2 via Centura SQL Network
- DB2400 – DB2 on AS/400 via Client Access
- DB2ODBC – DB2 using the IBM ODBC driver
- DB2UNIX – DB2 UNIX driver
- ORACLE – Oracle via the OCI interface
- INFORMIX – Native Informix
- SYBASE – Native Sybase
- MICROSOFT – SQL Server via ODBC
- APPSRV – Used to indicate that the database name is actually an application server name
- **DBNAME** – Name of the database or alias
- **DBQ** – Used to combine values separated by ‘/’, APPNAME/DBTYPE/DBNAME. The APPNAME value and the following slash are dropped when not in 3-tier.
- **SERVER** – Name of the database server, used by Sybase and Informix
- **UID** – PeopleSoft operator ID
- **PWD** – Password associated with the PeopleSoft operator

The driver uses any information it retrieves from the ODBC.INI file or registry to augment the information passed to it in the connection string. If the information in the ODBC.INI file or registry duplicates information in the connection string, the driver uses the information in the connection string.

The existing PeopleSoft database connection dialog will be used to prompt the user for any required connection information.

Information Procedures

ODBC defines these functions as a means for the application to get information about the ODBC driver and data source. Typically, the application calls these functions passing a value of the particular type of information of interest. These values are numerous and are defined in the Microsoft ODBC SDK Reference manual.

SQLGetInfo

SQLGetInfo returns general information about the driver and data source associated with an hdbc.

```
RETCODE SQLGetInfo(hdbc, fInfoType, rgbInfoValue, cbInfoValueMax, pcbInfoValue)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HDBC	Input	Connection handle.
2	UWORD	Input	Type of information. flInfoType must be a value representing the type of interest
3	PTR	Output	Pointer to storage for the information. Depending on the flInfoType requested
4	SWORD	Input	Maximum length of the rgbInfoValue buffer.
5	SWORD FAR *	Output	The total number of bytes (excluding the null termination byte for character data) available to return in rgbInfoValue.

SQLFunctions

SQLGetFunctions returns information about whether a driver supports a specific ODBC function.

```
RETCODE SQLGetFunctions(hdbc, fFunction, pfExists)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HDBC	Input	Connection handle.
2	UWORD	Input	SQL_API_ALL_FUNCTIONS or a #define value that identifies the ODBC function of interest.
3	UWORD FAR *	Output	If fFunction is SQL_API_ALL_FUNCTIONS, pfExists points to a UWORD array with 100 elements. The array is indexed by #define values used by fFunction to identify each ODBC function; some elements of the array are unused and reserved for future use. An element is TRUE if it identifies an ODBC function supported by the driver. It is FALSE if it identifies an ODBC function not supported by the driver or does not identify an ODBC function.

SQLGetTypeInfo

SQLGetTypeInfo returns information about data types supported by the data source. The driver returns the information in the form of an SQL result set.

```
RETCODE SQLGetTypeInfo(hstmt, fSqlType)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HSTMT	Input	Statement handle for the result set.
2	SWORD	Input	The SQL data type

Catalog Procedures (Meta data)

ODBC Listing procedures supply the client with Catalog table information. The PeopleSoft ODBC driver will support listings of queries and columns utilizing PeopleSoft metadata.

SQLProcedures

SQLProcedures returns the list of procedure names stored in a specific data source. Procedure is a generic term used to describe executable objects, or named entities that can be invoked using input and output parameters, and which can return result sets similar to the results returned by SQL SELECT statements.

```
RETCODE SQLProcedures(hstmt, szProcQualifier, cbProcQualifier, szProcOwner,
cbProcOwner, szProcName, cbProcName)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HSTMT	Input	Statement handle.
2	UCHAR FAR *	Input	Procedure qualifier.
3	SWORD	Input	Length of szProcQualifier.
4	UCHAR FAR *	Input	String search pattern for procedure owner names.
5	SWORD	Input	Length of szProcOwner.
6	UCHAR FAR *	Input	String search pattern for procedure names.
7	SWORD	Input	Length of szProcName.

This function is typically used before statement execution to retrieve information about procedures available from the data source's catalog.

SQLProcedures returns the results as a standard result set (when SQLFetch is called), ordered by PROCEDURE_QUALIFIER, PROCEDURE_OWNER, PROCEDURE_NAME, PROCEDURE_REMARKS and PROCEDURE_TYPE. The table below list the columns in the PeopleSoft result set.

<i>Column Name</i>	<i>Data Type</i>	<i>Description</i>
PROCEDURE_QUALIFIER	SQL_CHAR(128)	''
PROCEDURE_OWNER	SQL_CHAR(128)	'QUERY'

PROCEDURE_NAME	SQL_CHAR(128)	Query name with punctuation and spaces converted to underscore
REMARKS	SQL_CHAR(256)	Description of the Query, currently unused
PROCEDURE_TYPE	SQL_INT	SQL_PT_PROCEDURE

SQLProcedureColumns

```
RETCODE SQLProcedureColumns(hstmt, szProcQualifier, cbProcQualifier,
szProcOwner, cbProcOwner, szProcName, cbProcName, szColumnName, cbColumnName)
```

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle
2	UCHAR FAR*	Input	Procedure qualifier name.
3	SWORD	Input	Length of szProcQualifier.
4	UCHAR FAR*	Input	String search pattern for procedure owner names.
5	SWORD	Input	Length of szProcOwner.
6	UCHAR FAR*	Input	String search pattern for procedure names.
7	SWORD	Input	Length of szProcName.
8	UCHAR FAR*	Input	String search pattern for column names.
9	SWORD	Input	Length of szColumnName.

SQLProcedureColumns returns a list of input and output parameters, as well as the columns that make up the result set for the specified procedures. The driver returns the information as a result set on the specified hstmt.

This function is typically used before statement execution to retrieve information about procedure parameters and columns from the data source's catalog.

SQLProcedureColumns returns the results as a standard result set (when SQLFetch is called), ordered by PROCEDURE_QUALIFIER, PROCEDURE_OWNER, PROCEDURE_NAME, and COLUMN_TYPE. The table below lists the columns in the result set.

Column Name	Data Type	Description
PROCEDURE_QUALIFIER	SQL_CHAR(128)	N/A
PROCEDURE_OWNER	SQL_CHAR(128)	N/A

PROCEDURE_NAME	SQL_CHAR(128)	Procedure identifier
COLUMN_NAME	SQL_CHAR(128)	Procedure column identifier.
COLUMN_TYPE	SQL_INT	SQL_PARAM_INPUT or SQL_RESULT_COL
DATA_TYPE	SQL_INT	SQL data type
TYPE_NAME	SQL_CHAR(128)	Data type name of procedure column.
PRECISION	SQL_INT	Precision of the procedure column.
LENGTH	SQL_INT	Length in bytes of data transferred on an SQLGetData or SQLFetch operation.
SCALE	SQL_INT	Scale of procedure column.
RADIX	SQL_INT	N/A
NULLABLE	SQL_INT	Whether the procedure column accepts a NULL value.
REMARKS	SQL_CHAR(256)	A description of the procedure column.

PeopleSoft Driver

The driver will return information for the first query requested only. It will not return results for multiple queries. The driver will call the new query API functions *QpmDescribeParm* and *QpmDescribeCol*. *QpmDescribeParm* will walk the query definition stored in *hstmt* and for each prompt variable return a *SQLProcedureColumns* result row of COLUMN_TYPE equal to *SQL_PARAM_INPUT*. *QpmDescribeCol* will walk the same query definition and for each result column return a *SQLProcedureColumns* result row of COLUMN_TYPE equal to *SQL_RESULT_COL*. The *szProcQualifier* and *szProcOwner* criterion will be ignored. The result set returned will be for the current *OprId*. The result set columns for Procedure Qualifier and Procedure Remarks do not apply and are set to NULL with a 1 byte column length. The Procedure Owner column is set to either the *OprId* or Public.

Executing SQL

The minimum ODBC SQL conformance level requires the driver to support the following :

- Data Definition Language (DDL): **CREATE TABLE** and **DROP TABLE**.
- Data Manipulation Language (DML): simple SELECT, INSERT, UPDATE SEARCHED, and DELETE SEARCHED.
- Expressions: simple (such as $A > B + C$).
- Data types: CHAR, VARCHAR, or LONG VARCHAR.

The PeopleSoft ODBC driver will **not** support the minimum SQL conformance level even though it reports supporting extended syntax. Open Query will only support the ODBC extended SQL Grammar for stored procedures. The stored procedure syntax is:

```
{[? = ] call procedure_name [ (param, ...)]}
```

The stored procedure execution model was chosen because it supports the independent creation of an SQL statement. In our case, the independent creation is done through PeopleSoft Query. However, instead of a stored procedure, the result is a PeopleSoft Query object.

Statement Handle

Before an application can execute an SQL statement, it must allocate a statement handle for the statement. To allocate a statement handle, an application declares a variable of type HSTMT and passes its address to SQLAllocStmt.

SQLAllocStmt

```
RETCODE SQLAllocStmt(hdbc, phstmt)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HDBC	Input	Connection handle.
2	HSTMT FAR*	Output	Pointer to storage for the statement handle.

SQLAllocStmt allocates memory for a statement handle and associates the statement handle with the connection specified by *hdbc*.

A statement handle references statement information, such as network information, SQLSTATE values and error messages, cursor names, number of result set columns and status information for SQL statement processing.

If the application calls *SQLAllocStmt* with a pointer to a valid *hstmt*, the driver overwrites the *hstmt* without regard to its previous contents.

Execution Models

ODBC supports three execution models. Each accomplishes the same tasks, but each one differs with regard to when and where (on the client or on the server) each step is performed.

ExecDirect

In this model, the SQL statement is specified, sent to the server, and executed, all in one step. This model is best suited for ad hoc SQL statements or SQL statements that will be executed only once. Parameters can be used, but they act merely as place-holders that the driver replaces with the parameter values before it sends the SQL statement to the server.

The DBMS discards the optimization information used to execute the SQL statement after execution is complete. If the same statement is specified again with *SQLExecDirect*, the entire process of parsing and optimizing happens again.

Prepare/Execute

In this model, the SQL statement is “prepared” (sent to the server, parsed, and optimized) first and executed later. When the statement is executed, what flows to the server is not the SQL statement itself, but merely some way of referencing the statement so that the access plan can be executed immediately. Parameters are often used in these SQL statements, so the only items that flow to the server are the reference to the access plan and the parameter values, not the entire SQL statement.

The Prepare/Execute model should be used when repeated execution of the same SQL statement is needed and when the SQL statement must be constructed dynamically during runtime. To use this model, the application calls *SQLPrepare*, and then (presumably in a loop) calls *SQLExecute*.

Stored Procedures

The stored procedure model is like the Prepare/Execute model except that with stored procedures, the preparation step can be done independently from the application and the stored procedure persists beyond the runtime of the application. To use stored procedures in ODBC, the application calls *SQLExecDirect* but uses the SQL statement to specify the stored procedure name:

```
SQLExecDirect(hstmt, "{call query.procl(?,?,?)}", SQL_NTS);
```

SQLExecDirect

```
RETCODE SQLExecDirect(hstmt, szSqlStr, cbSqlStr)
```

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle
2	UCHAR FAR*	Input	SQL statement to be executed.
3	SDWORD	Input	Length of szSqlStr.

SQLExecDirect executes a preparable statement, using the current values of the parameter marker variables if any parameters exist in the statement. The application calls *SQLExecDirect* to send an SQL statement to the data source. The driver modifies the statement to use the form of SQL used by the data source, then submits it to the data source. In particular, the driver modifies the escape clauses used to define ODBC-specific SQL grammar extensions.

The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) into the SQL statement at the appropriate position. It is unnecessary to use any parameter markers, as PeopleSoft Query objects know the exact number of prompt values. The PeopleSoft driver will prompt the user for input

values at this time, if no values were previously supplied via the input or the `SQLBindParameter` function.

Only stored procedures (predefined queries) are supported.

In addition to the ODBC error conditions, the PeopleSoft driver will return an error condition if the following conditions are true:

- A valid PeopleSoft query name can not be found or loaded
- Prompt values can not be satisfied via a prompting page

SQLPrepare

RETCODE **SQLPrepare**(hstmt, szSqlStr, cbSqlStr)

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle
2	UCHAR FAR*	Input	SQL statement to be executed.
3	SDWORD	Input	Length of szSqlStr.

SQLPrepare prepares an SQL string for execution. The application calls *SQLPrepare* to send an SQL statement to the data source for preparation. The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) into the SQL string at the appropriate position. Once a statement is prepared, the application uses *hstmt* to refer to the statement in later function calls. The prepared statement associated with the *hstmt* may be re-executed by calling *SQLExecute* until the application frees the *hstmt* with a call to *SQLFreeStmt* with the `SQL_DROP` option or until the *hstmt* is used in a call to *SQLPrepare*, *SQLExecDirect*, or one of the catalog functions (*SQLColumns*, *SQLTables*, and so on). Once the application prepares a statement, it can request information about the format of the result set.

Only stored procedures (predefined queries) are supported.

SQLExecute

RETCODE **SQLExecute**(hstmt)

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle

SQLExecute executes a prepared statement, using the current values of the parameter marker variables if any parameter markers exist in the statement. *SQLExecute* executes a statement prepared by *SQLPrepare*. Once the application processes or discards the results from a call to *SQLExecute*, the application can call *SQLExecute* again with new parameter values.

To execute a SELECT statement more than once, the application must call *SQLFreeStmt* with the SQL_CLOSE parameter before reissuing the SELECT statement.

As in the SQLExecDirect function, the PeopleSoft ODBC driver will prompt the user for input values if values have not been previously supplied.

Descriptive Information

In order for applications to be flexible enough for ad-hoc SQL, it is necessary to provide the application a means to query the ODBC driver for information pertaining to required storage and data types. This is done via descriptive functions defined by the ODBC specification. ODBC enabled applications use these functions to dynamically query the driver for information about the result set and the input and output values.

SQLColAttributes

```
RETCODE SQLColAttributes(hstmt, icol, fDescType, rgbDesc, cbValueMax, pcbValue)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HSTMT	Input	Statement handle
2	UWORD	Input	Column number of result data.
3	UWORD	Input	Valid descriptor type.
4	PTR	Output	Pointer to storage for descriptor information
5	SWORD	Input	Maximum buffer size
6	SWORD FAR*	Output	Output length of data in buffer.
7	SWORD FAR*	Output	Pointer to integer output data.

SQLColAttributes returns descriptor information for a column in a result set; it cannot be used to return information about the bookmark column (column 0). Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

SQLDescribeCol

```
RETCODE SQLDescribeCol(hstmt, icol, szColName, cbColNameMax, pcbColName,
    pfSqlType, pcbColDef, pibScale, pfNullable )
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HSTMT	Input	Statement handle
2	UWORD	Input	Column number of result data.
3	UCHAR FAR*	Output	Pointer to storage for the column name.

4	SWORD	Input	Maximum length of the szColName buffer.
5	SWORD FAR*	Output	Total number of bytes available to return in szColName.
6	SWORD FAR*	Output	The SQL data type of the column.
7	UDWORD FAR*	Output	The precision of the column on the data source.
8	SWORD FAR*	Output	The scale of the column on the data source.
9	SWORD FAR*	Output	Indicates whether the column allows NULL values.

SQLDescribeCol returns the result descriptor - column name, type, precision, scale, and nullability - for one column in the result set. An application typically calls *SQLDescribeCol* after a call to *SQLPrepare* and before or after the associated call to *SQLExecute*. An application can also call *SQLDescribeCol* after a call to *SQLExecDirect*.

SQLDescribeParam

```
RETCODE SQLDescribeParam(hstmt, ipar, pfSqlType, pcbColDef, pibScale,
pfNullable)
```

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle
2	UWORD	Input	Marker number.
3	SWORD FAR*	Output	Pointer to storage for the SQL type.
4	SWORD FAR*	Output	Pointer to storage for precision of value.
5	SWORD FAR*	Output	Pointer to storage for scale of value.
6	UDWORD FAR*	Output	Pointer to storage for nullable flag.

SQLDescribeParam returns the description of a parameter marker associated with a prepared SQL statement. In terms of PeopleSoft Query objects, this is the description of the prompt values required to fulfill the query keys.

SQLGetRowCount

```
RETCODE SQLRowCount(hstmt, pcrow)
```

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle.

2	SDWORD FAR *	Input	Pointer to storage of the row counter.
---	--------------	-------	--

SQLRowCount returns the number of rows affected by an **UPDATE**, **INSERT**, or **DELETE** statement or by a SQL_UPDATE, SQL_ADD, or SQL_DELETE operation in *SQLSetPos*. If called during a fetch cycle, the value returned is the number of rows returned to the application at the current position.

SQLNumParams

RETCODE **SQLNumParams**(hstmt, pccol)

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle
2	SWORD FAR*	Output	Number of parameters in the statement.

SQLNumParams returns the number of parameters in an SQL statement.

SQLNumResultCols

RETCODE **SQLNumResultCols**(hstmt, pccol)

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle
2	SWORD FAR*	Output	Number of columns in the result set.

SQLNumResultCols returns the number of columns in the result set. *SQLNumResultCols* can be called successfully only when the hstmt is in the prepared or executed state. An application typically would use value returned in *pccol* in a loop and call *SQLDescribeCol* for each column in the result set.

Binding Application Data

An application retrieves an entire row of values using a technique called binding. Binding associates the data from the data source with variables in the application program. There are two directions of binding, input and output. Input data must always be bound. On output, once a result column is bound, the variable receives the value for that column each time a new row is fetched. The following example shows how this technique differs from *SQLGetData*.

```
/* for all columns in the current result set */
for (i = 0; i < columns; i++)
    SQLBindCol(hstmt, ..., &value[i], ...)

while (SQL_SUCCESS == (rc = SQLFetch(hstmt)))
    /* value[ i .. n] contains data for current row */
```

SQLBindCol

```
RETCODE SQLBindCol(hstmt, icol, fCType, rgbValue, cbValueMax, pcbValue)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HSTMT	Input	Statement handle
2	UWORD	Input	Column number of result data.
3	SWORD	Input	The C data type of the result data.
4	PTR	Both	A pointer to storage for the result column.
5	SDWORD	Input	Maximum length of the rgbValue buffer.
6	SDWORD FAR*	Both	A pointer to a buffer for the SQL_NULL_DATA or the number of bytes available to return in rgbValue prior to calling SQLFetch.

SQLBindCol assigns the storage and data type for a column in a result set.

SQLBindParameter

```
RETCODE SQLBindParameter(hstmt, ipar, fParamType, fCType, fSqlType, cbColDef,
ibScale, rgbValue, cbValueMax, pcbValue)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HSTMT	Input	Statement handle
2	UWORD	Input	Parameter number, ordered sequentially left to right, starting at 1.
3	SWORD	Input	The type of the parameter.
4	SWORD	Input	The C data type of the parameter.
5	SWORD	Input	The SQL data type of the parameter.
6	UDWORD	Input	The precision of the column or expression of the corresponding parameter marker.
7	SWORD	Input	The scale of the column or expression of the corresponding parameter marker
8	PTR	Both	A pointer to a buffer for the parameter's data.
9	SDWORD	Input	Maximum length of the rgbValue buffer.

10	SDWORD FAR*	Both	A pointer to a buffer for the parameter's length.
----	-------------	------	---

SQLBindParameter binds a buffer to a parameter marker in an SQL statement. An application calls *SQLBindParameter* to bind each parameter marker in an SQL statement. Bindings remain in effect until the application calls *SQLBindParameter* again or until the application calls *SQLFreeStmt* with the SQL_DROP or SQL_RESET_PARAMS option.

An application can use *SQLBindParameter* to supply the prompt values for a PeopleSoft query. *SQLBindParameter* will call the new function, *ODBCBindParm*. *ODBCBindParm* will convert the ODBC C data type, *fCType*, to the ODBC SQL data type, *fSqlType*. It will then map the ODBC SQL data type to a supported PeopleSoft RDM data type and call the appropriate internal bind function. Appendix B shows the data types supported by the PeopleSoft driver.

An ODBC driver is required to support conversions from all ODBC C data types to the ODBC SQL data types that they support.

Literal Parameters

An application may also supply prompt values as literal strings embedded in the SQL statement string. For example:

```
SQLExecDirect(hstmt, "{call query.myquery(8001, NEWGN)}", SQL_NTS);
```

If prompt values are not provided, Query displays a dialog to prompt the user for each required value at the time of statement execution.

Retrieving Results

For row-returning statements such as SELECTs or stored procedures, ODBC provides three ways to retrieve data. Using a single function call, an application can retrieve a single value, an entire row of values, or multiple rows of values. The PeopleSoft driver only supports the first two methods; single value and entire row.

Retrieving One Value Directly

One way an application can retrieve data is by using a function call (*SQLGetData*) for every column in every row. The application supplies function arguments that specify the column number and a variable in which to receive the data, and after the function call has been successfully executed the value for the given column is returned in the variable. The application uses two loops to retrieve an entire result set, as in this example:

```
/* For all rows */
while (SQL_SUCCESS == (rc = SQLFetch(hstmt)))
    /* for all columns in current results set */
    for (colnum = 1; colnum <= columns; colnum++)
        SQLGetData(hstmt, colnum, ..., &value, ..)
```

SQLGetData is also used for another purpose: the piece-meal retrieval of large text and binary data (such as images). It is often difficult or impossible for an application to allocate a single piece of memory big enough to hold a large data object, such as a 50-page document or a high-density bitmap.

SQLFetch

RETCODE **SQLFetch**(hstmt)

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle

SQLFetch fetches a row of data from a result set. The driver returns data for all columns that were bound to storage locations with *SQLBindCol*. *SQLFetch* positions the cursor on the next row of the result set. When the cursor is positioned to the last row of the result set, *SQLFetch* returns SQL_NO_DATA_FOUND.

If the application called *SQLBindCol* to bind columns, *SQLFetch* stores data into the locations specified by the calls to *SQLBindCol*. If the application does not call *SQLBindCol* to bind any columns, *SQLFetch* doesn't return any data; it just moves the cursor to the next row. An application can call *SQLGetData* to retrieve data not previously bound to a storage location.

SQLGetData

RETCODE **SQLGetData**(hstmt, icol, fCType, rgbValue, cbValueMax, pcbValue)

Argument	Type	Use	Description
1	HSTMT	Input	Statement handle
2	UWORD	Input	Column number of result data.
3	SWORD	Input	The C data type of the result data.
4	PTR	Both	A pointer to storage for the result column.
5	SDWORD	Input	Maximum length of the rgbValue buffer.
6	SDWORD FAR*	Both	A pointer to a buffer for the SQL_NULL_DATA or the number of bytes available to return in rgbValue prior to calling <i>SQLFetch</i> .

SQLGetData returns result data for a single unbound column in the current row. The application must call *SQLFetch* to position the cursor on a row of data before it calls *SQLGetData*. It is possible to use *SQLBindCol* for some columns and use *SQLGetData* for others within the same row. This function can be used to retrieve character or binary data values in parts from a column

with a character, binary, or data source specific data type (for example, data from SQL_LONGVARBINARY or SQL_LONGVARCHAR columns).

Retrieving Status and Error Information

When any ODBC call fails, the driver is responsible for retaining the error information until the next ODBC call. The error state and error text is retrieved from the driver with the *SQLError* function.

SQLError

```
RETCODE SQLError(henv, hdbc, hstmt, szSqlState, pfNativeError,
szErrorMsg, cbErrorMsgMax, pcbErrorMsg)
```

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HENV	Input	Environment handle or SQL_NULL_HENV.
2	HDBC	Input	Connection handle or SQL_NULL_HDBC.
3	HSTMT	Input	Statement handle or SQL_NULL_HSTMT.
4	UCHAR FAR *	Output	SQLSTATE as null terminated string.
5	SDWORD FAR *	Output	Native error code (specific to the data source).
6	UCHAR FAR *	Output	Pointer to storage for the error message text.
7	SWORD	Input	Maximum length of the szErrorMsg buffer. This must be less than or equal to SQL_MAX_MESSAGE_LENGTH - 1.
8	SWORD FAR *	Output	Pointer to the total number of bytes (excluding the null termination byte) available to return in szErrorMsg. If the number of bytes available to return is greater than or equal to cbErrorMsgMax, the error message text in szErrorMsg is truncated to cbErrorMsgMax - 1 bytes.

SQLError returns error or status information. An application typically calls *SQLError* when a previous call to an ODBC function returns SQL_ERROR or SQL_SUCCESS_WITH_INFO. The application can, however, call *SQLError* after any ODBC function call.

Terminating Transactions and Connections

Each query object that runs in ODBC creates a transaction. In order to ensure that all memory associated with a transaction is freed and locks released, an application should call `SQLTransact`.

SQLTransact

RETCODE `SQLTransact` (*henv*, *hdbc*, *fType*)

Argument	Type	Use	Description
1	HENV	Input	Environment handle or SQL_NULL_HENV.
2	HDBC	Input	Connection handle or SQL_NULL_HDBC.
3	UWORD	Input	Flag for SQL_COMMIT or SQL_ROLLBACK.

`SQLTransact` requests a commit or rollback operation for all active operations on all *hstmts* associated with a connection. `SQLTransact` can also request that a commit or rollback operation be performed for all connections associated with the *henv*. In the case of query objects, the transaction is automatically closed upon termination of the statement handle.

SQLDisconnect

RETCODE `SQLDisconnect` (*hdbc*)

Argument	Type	Use	Description
1	HDBC	Input	Connection handle

`SQLDisconnect` closes the connection associated with a specific connection handle.

If an application calls `SQLDisconnect` before it has freed all *hstmts* associated with the connection, the driver frees those *hstmts* after it successfully disconnects from the data source. However, if one or more of the *hstmts* associated with the connection are still executing asynchronously, `SQLDisconnect` will return `SQL_ERROR`.

SQLFreeConnect

RETCODE `SQLFreeConnect` (*hdbc*)

Argument	Type	Use	Description
1	HDBC	Input	Connection handle

SQLFreeConnect releases a connection handle and frees all memory associated with the handle. This is called after *SQLDisconnect*.

SQLFreeEnv

RETCODE **SQLFreeEnv** (henv)

<i>Argument</i>	<i>Type</i>	<i>Use</i>	<i>Description</i>
1	HENV	Input	Environment handle.

SQLFreeEnv frees the environment handle and releases all memory associated with the environment handle.

ODBC Compliance

The ODBC API defines a set of core functions that correspond to the functions in the X/Open and SQL Access Group Call Level Interface specification. ODBC also defines two extended sets of functionality, Level 1 and Level 2.

For the specific ODBC API descriptions and implementation details, see Microsoft® Open Database Connectivity™ Software Development Kit, Version 2.50, For the Microsoft Windows™ and Windows NT™ Operating Systems ©1992, 1993, 1994, 1995, 1996, 1997 Microsoft Corporation. All rights reserved.

The following list summarizes the functionality included in each conformance level.

Core API

- Allocate and free environment, connection and statement handles.
- Convert to data sources. Use multiple statements on a connection.
- Prepare and execute SQL statements. Execute SQL statements immediately.
- Assign storage for parameters in an SQL statement and result columns.
- Retrieve data from a result set. Retrieve information about a result set.
- Commit or rollback transactions.
- Retrieve error information.

Level 1 API

- Core API functionality.

- Connect to data sources with driver-specific dialog boxes.
- Set and inquire values of statement and connection options.
- Send part or all of a parameter value (useful for long data).
- Retrieve part or all of a result column value (useful for long data).
- Retrieve catalog information (columns, special columns, statistics, and tables).
- Retrieve information about driver and data source capabilities, such as supported data types, scalar functions, and ODBC functions.

Level 2 API

- Core and Level 1 API functionality
- Browse connection information and list available data sources.
- Send arrays of parameter values. Retrieve arrays of result columns values.
- Retrieve the number of parameters and describe individual parameters.
- Use a scrollable cursor.
- Retrieve the native form of an SQL statement
- Retrieve catalog information (privileges, keys, and procedures).
- Call a translation DLL.

To claim that it conforms to a given API or SQL conformance level, a driver must support all the functionality in that conformance level, regardless of whether that functionality is supported by the DBMS associated with the driver. However, conformance levels do not restrict drivers to the functionality in the levels to which they conform. Drivers can support as much functionality as they can; applications can determine the functionality supported by a driver by calling *SQLGetInfo*, *SQLGetFunctions*, and *SQLGetTypeInfo*.

ODBC to RDM Data Types

The following table shows the mapping from ODBC C data types to ODBC SQL and PeopleSoft RDM data types.

<i>RDM Type</i>	<i>fSqlType</i>	<i>Type</i>
RDM_CHAR	SQL_CHAR	unsigned char FAR*
RDM_LONG_CHAR	SQL_VARCHAR	unsigned char FAR*
RDM_NUMBER, RDM_SIGNED_NUMBER	SQL_NUMERIC	unsigned char FAR*

RDM_DATE	SQL_DATE	struct tag DATE_STRUCT { UWORD year; UWORD month; UWORD day; }
RDM_TIME	SQL_TIME	struct TIME_STRUCT { UWORD hour; UWORD minute; UWORD second; }
RDM_DATETIME	SQL_TIMESTAMP	struct TIMESTAMP_STRUCT { SWORD year; UWORD month; UWORD day; UWORD hour; UWORD minute; UWORD second; UWORD fraction; }
RDM_IMAGE	SQL_LONGVARBINARY	unsigned char FAR *

Example Using the Open Query ODBC API

The following example shows the ODBC API calls needed to execute a PeopleSoft Query using the PeopleSoft Open Query ODBC driver. The query “myquery” requires two bind variables: business unit and department id. “myquery” returns an answer set of 3 columns: Employee ID, Name and Monthly Rate.

```

/*****
* Function:      OpenQuerySample
*
* Description:   Sample program illustrating the usage of PeopleSoft's
*               Open Query ODBC API.
*               Sample code uses basic PeopleSoft Query ODBC interface
*               functions. Most error checking is excluded to make
*               code easier to follow; in a typical application,
*               every return code would be checked.
*
* Returns:      TRUE if successful
*****/

BOOL WINAPI      OpenQuerySample(HWND hWnd)
{
    HENV          hEnv;           // Environment handle for application
    HDBC          hDbc;           // Connection handle
    HSTMT         hStmt;          // Statement handle
    RETCODE       retcode;        // Return code
    char          szConnectString[] =

```

```

        "DSN=PeopleSoft PeopleTools;DBTYPE=ORACLE;DBNAME=PTDMO7;UID=PTDMO;PWD=PTDMO;";
char          szConnStringOut[256];    // completed connection string
WORD          nConnStringLength;      // length of completed connect string

// Allocate environment, database connection
retcode = SQLAllocEnv(&hEnv);
if ((retcode = SQLAllocConnect(hEnv, &hDbc)) != SQL_SUCCESS)
    // error--this would normally abort program with message
    return(FALSE);

// Connect to the database
retcode = SQLDriverConnect(hDbc, hWnd, szConnectString,
    strlen(szConnectString), szConnStringOut, sizeof(szConnStringOut),
    &nConnStringLength, SQL_DRIVER_COMPLETE);

retcode = SQLAllocStmt(hDbc, &hStmt);

ProcessQuery(hStmt);

// Close the connection, release resources
retcode = SQLFreeStmt(hStmt);
retcode = SQLDisconnect(hDbc);
retcode = SQLFreeConnect(hDbc);
retcode = SQLFreeEnv(hEnv);

return(TRUE);
}

/*****
* Function:      ProcessQuery
*
* Description:   Run a query and retrieve answer set.
*
* Returns:      TRUE if successful, else FALSE
*****/

BOOL          ProcessQuery(HSTMT hStmt)
{
    RETCODE     retcode;                // Return code
    char        szSelect[] = "{call query.myquery(?,?)";

    // binding of input variables must occur before statement execution
    for (i = 0; i < 2; i++)
        retcode = SQLBindCol(hStmt, i, datatype, &value, sizeof(value), &valueLen);

    retcode = SQLExecDirect(hStmt, szSelect, strlen(szSelect));

    while (retcode = SQLFetch(hStmt) == SQL_SUCCESS)
        // process data for a fetched row....

    return(retcode == SQL_NO_DATA_FOUND);
}

```


CHAPTER 6

PeopleTools Command Line Parameters

The simplest way to link a PeopleSoft application and a third-party application is to launch the PeopleTools executable file from the third-party application using a WinEXE-type function. This method enables you to start a PeopleSoft application or PeopleTools, although it does not establish any kind of connection between the applications.

PeopleTools offers a variety of command-line parameters that you can use to control what database it connects to and what window or page it displays. Using these parameters, you can automatically navigate to the part of the system the user needs.

This section lists the command line parameters you can use to start PeopleTools.

Command Line for Start

The command line for starting PeopleSoft applications or PeopleTools has the following syntax:

```
PSTOOLS [-parameter value [-parameter value . . .]]
```

The command line for starting Data Mover has the following syntax:

```
PSDMT [-parameter value [-parameter value . . .]]
```

You can include as many or as few parameters as you need or none.

Each parameter starts with a hyphen (-) or a forward slash (/). The value for each parameter follows it, separated by zero or more spaces. In general, the value doesn't need to have quotes around it, even if it has internal spaces—the system treats all text following the parameter as part of the value, up to the next parameter or the end of the command line.



You need to enclose a value in double quotes only when it includes a hyphen or forward slash, or when you want to include leading or trailing spaces. If the value itself includes a double quote character, precede the double quote with a back slash (\).

If you pass incorrect values, or if the specified User doesn't have security access to the specified menu or page, the system returns the error "You are not authorized to access this component. (40,20)."

Command Line Parameters

The available parameters for PSTOOLS.EXE fall into three general categories:

- Parameters providing login information
- Parameters specifying which window or page to display
- Parameters setting general options

All of the available parameters are listed in the table below.

If the command line includes login parameters, it uses them only if no PeopleSoft applications are currently running. If you already have a PeopleSoft application running, the system starts a new instance using the same login information as the active instance.

Parameter	Value	Description
-CT	Database type	The type of database to connect to. The valid values are: SQLBASE, ORACLE, INFORMIX, SYBASE, MICROSOFT, DB2, DB2ODBC, DB2MDI, DB2400, DB2UNIX (Note the spelling of MICROSOFT).
-CS	Server name	The name of the database server for the database you're connecting to. This setting is required for some database types.
-CD	Database name	The name of the database to connect to, as you would enter it into the PeopleSoft Login dialog box.
-CO	User ID	The PeopleSoft User ID to use to login.
-CP	Password	The password for the specified User ID.
-CI	Connect ID	The ID used to connect to the database server.
-CW	Password	The password for the specified Connect ID.
-SS	NO	Suppresses the display of the PeopleSoft splash screen.
-SN	NO	Suppresses the sound that plays when you log onto the PeopleSoft system.
-MN	Menu name	The name of the window to start. Use the PSMENUITEM.MENUNAME value, such as "MAINTAIN_SALES_ORDERS". Use the menu name as it appears in the Application Designer, <i>not</i> the menu display text.
-MB	Menu bar name	The menu bar to start. Use the PSMENUITEM.BARNAME value, such as "USE_AM". Use the bar name, <i>not</i> the bar label.
-MI	Menu item name	The menu item to select. Use the PSMENUITEM.ITEMNAME value, such as "HEADER". Use the item name, <i>not</i> the label.

Parameter	Value	Description
-MP	Page name	<p>The page to start. Use the PSPNLGROUP.ITEMNAME value, such as “GENERAL_1”. Use the page name, <i>not</i> its display text.</p> <p>You need to provide the page name even if it’s the same as the menu item name.</p>
-MA	Action mode	<p>The action mode to use for the selected page. Which actions are available depends on the page.</p> <p>Use the text of the action mode as it appears on the menu, including an ampersand (&) before the underlined letter. The complete set is:</p> <p>&Add, &Update/Display, Update/&Display All, &Correction, Data &Entry</p> <p>The default is &Update/Display, if that action is available for the selected page.</p>
-K	Key list	<p>A list of key field/value pairs for the record definition underlying the selected page. The format of each pair is:</p> <p><i>fieldname=value</i></p> <p>For example, “-K SETID=MFG -K CUST_ID=NEXT”</p> <p>You should provide a value for each key field at level 0 on the selected page.</p>
-W	None	Display the Select Worklist dialog box for the current User ID.
-QUIET	(none)	Run in “quiet mode,” so that no message boxes appear.
-@	Filename	<p>Read the command line parameters from the specified text file. Include all the parameters as a single line. PSTOOLS.EXE uses all the parameters on the command line, plus all the parameters listed in the file. Essentially, PSTOOLS.EXE inserts the contents of the file at the location of the -@ parameter.</p>
-P	(none)	Delete the parameter files specified on the current command line with the -@ parameter.
-FP	Filename	Data Mover only. The name of the Data Mover script to run.

Examples

This command line starts PeopleTools, logs onto the DEP7TST database, opens the Maintain Customers window, and displays the General Information page from the Use menu. The user sees the page in Add mode with the Set ID and Cust ID already set:

```
PSTOOLS -CT MICROST -CS SEPNTDB05 -CD EP7TST -CO VP1 -CP VP1
-MN MAINTAIN_CUSTOMERS -MB USE -MI GENERAL_INFORMATION -MP GENERAL_1
-MA &Add -K SETID=MFG -K CUST_ID=NEXT
```

The next example shows a command line that takes input from a text file named PSRMA.TXT:

```
PSTOOLS -@PSRMA.TXT
```

The text file contains a single line of text giving the command line parameters:

```
-CTMICROST -CSSEPNTDB05 -CDEP7TST -COVP1 -CPVP1
-MNMANAGE_RETURNED_MATERIAL -MBUSE -MIRMA_FORM -MPRMA_FORM2
-MA&Update/Display -KBUSINESS_UNIT=M04A -KRMA_ID=EW100
```

The last example includes a menu bar name with an internal hyphen. That value must be surrounded by double quotes:

```
PSTOOLS -CT MICROST -CS SEPNTDB05 -CD EP7TST -CO VP1 -CP VP1
-MN MAINTAIN_SALES_ORDERS -MB "USE_A-M" -MI HEADER -MP SOLD_TO
-MA&Add
```

Command Line for Project Build

The command line statement for the project build does the following:

- Connects the project to the source database
- Uses the build settings from the Windows registry to generate the SQL script for create or alter

Verify the following before executing the build command line:

- Create and load system components of the project
- Set the build settings in the Windows registry

Option	Description	Error Handling
-CT	Database Type. The type of database to connect to. (MICROST, ORACLE, SYBASE, and so on)	Required. If this parameter is not supplied, the last database type is taken from the registry. If it fails, further execution is stopped and the error messages are written to the log file.
-CS	Server name. The name of the source database server for the database to which you are connecting. This setting is required for some database types.	Required for some database types. If this parameter is not supplied, further execution is stopped and the error messages are written to the log.

-CD	Database name. The name of the source database to connect to, as you would enter it into the PeopleSoft Login dialog box.	Required. If this parameter is not supplied, further execution is stopped and the error messages are written to the log file.
-CO	User ID. Peoplesoft User ID to login to source.	Required. If this parameter is not supplied, the last database type is taken from the registry. If it fails, further execution is stopped and the error messages are written to the log file.
-CP	User Password. The password for the specified User ID for source.	Required. If this parameter is not supplied the PeopleSoft Login dialog is prompted for the user to give the valid User password. If it fails, further execution is stopped and the error messages are written to the log file.
-PJB	Project Name. The name of the project which is to be built. This project should be available in the database before starting the command line project build.	Required. This is the main parameter which is used internally by the EXE to decide that the user the trying to do project build. If this parameter is not specified and if all the source login parameters are given, this EXE will just launch the application.

Example

Assume the following:

- Project name = CJR1
- Database type = Microsoft
- Database name = CJR810G
- User ID = PTDMO
- Password = PTDMO
- Pathname of SQL script file = c:\temp\psbuild.sql
- Pathname of log file = c:\temp\psbuild.log in the windows registry.

You enter the following command line:

```
PSTOOLS.EXE -CT MICROSOFT -CD CJR810G -CO PTDMO -CP PTDMO -PJB CJR_PRJ
```

Command Line for Upgrade Copy

The command line executable for the upgrade copy does the following:

- Connects to the source database
- Connects to the target database
- Copies the project and its objects from the source to target. If the same project already exists in the target database, it overwrites the same with the new project.

The details about the command line parameters are as follows:

Option	Description	Error Handling
-CT	Source – Database Type. The type of database to connect to source. (For example SQLBASE, ORACLE, SYBASE, and so on).	Required. If this parameter is not supplied, the database type is taken from the registry. If it fails, further execution is stopped and the error messages are written to the log file (if log file name parameter is specified).
-CS	Source – Server name. The name of the source database server for the database you're connecting to. This setting is required for some database types.	Required for some database types. If this parameter is not supplied, further execution is stopped and the error messages are written to the log file (if log file name parameter is specified).
-CD	Source – Database name. The name of the source database to connect to, as you would enter it into the PeopleSoft Login dialog box.	Required. If this parameter is not supplied, further execution is stopped and the error messages are written to the log file (if log file name parameter is specified).
-CO	Source – User ID. The Peoplesoft User ID to use to login to source.	Required. If this parameter is not supplied, the user id is taken from the registry. If it fails, further execution is stopped and the error messages are written to the log file (if log file name parameter is specified).
-CP	Source – User Password. The password for the specified User ID for source.	Required. If this parameter is not supplied, the PeopleSoft Login dialog prompts the user to give the valid User password. If it fails, further execution is stopped and the error messages

		are written to the log file (if log file name parameter is specified).
-TS	Target – Server name. The name of the target database server for the database you're connecting to. This setting is required for some database types.	Required for some database types. If this parameter is not supplied, further execution is stopped and the error messages are written to the log file (if log file name parameter is specified).
-TD	Target – Database name. The name of the target database to connect to, as you would enter it into the PeopleSoft Login dialog box.	Required. If this parameter is not supplied, further execution is stopped and the error messages are written to the log file (if log file name parameter is specified).
-TO	Target – User ID. The Peoplesoft User ID to use to login to the target.	Required. If this parameter is not supplied, further execution is stopped and the error messages are written to the log file (if log file name parameter is specified).
-TP	Target – User Password. The password for the specified User ID for the target.	Required. If this parameter is not supplied, further execution is stopped and the error messages are written to the log file (if log file name parameter is specified).
-LF	Log File name. The name of the file in which the error messages are logged to during the commandline upgrade copy process.	Not required. If this parameter is specified, a file is created in the specified path & name and all the error messages are written to that file.
-PJC	Source – Project Name. The name of the project that is to be copied from source to target. This project should be available in the source before starting the command line upgrade copy to target.	Required. This is the main parameter which is used internally by the EXE to identify that the user is trying to do upgrade copy. If this parameter is not specified and if all the source login parameters are given, this EXE will just launch the application.
-CL	Commit Limit (Number > 0). The Commit Limit for the number of objects copied/compared before a	Not required. The default will be 50 if the user does not set this parameter.

	commit is issued. Example: -CL 150	
-AF	Audit Flags on Records (Number -> 0 or 1). This indicates if the Target Audit Flags are kept or Set Target Audit Flags from Source. If the value is set 0, then the Target Audit Flags are taken from Source. If the value is set 1, then the Target Audit Flags are kept as is. Example: -AF 0	Not required. The default will be (1) to keep Target Audit Flags.
-DDL	DDL on Records and Indexes (Number -> 0 or 1). This indicates if the Target DDL flags are to be kept or Set Target DDL Flags from Source. If the value is set 0, then the Target DDL Flags are taken from Source. If the value is set 1, then the Target DDL Flags are kept as is. Example: -DDL 0	Not required. The default will be (1) to keep Target DDL Flags.
-OBJ	Object Type to Copy (Numbers with comma (,) as the delimiter). List of object types to Copy or All. There are 55 object types and please refer to the Object Type Selection Table given below while choosing the objects. For example: If you are choosing just Records and Indexes alone for copying, choose the appropriate numbers for Records and Indexes from the table below. For Records, the number is 0, and for the Indexes the number is 1. Example: -OBJ 0,1	Not required. If this parameter is not specified, then all of the objects will be copied by default.
-RST	Reset Done Flags (Number -> 0 or 1) This is to reset the done flags before initiating the copy. If the value is 0, then the done flags are not reset before initiating the copy. If the value is 1, then the done flags are reset before initiating the copy. Example: -RST 0	Not required. The default will be (1) to reset the done flags before initiating the copy.

-LNG	Copy Languages (Numbers with comma (,) as the delimiter) This is the list of Languages to copy. There are 10 languages and please refer to the Language Selection Table given below while choosing the languages. For example: If you are choosing just English and Spanish alone for copying, choose the appropriate codes for English and Spanish from the table below. For English, the code is ENG, and for Spanish the code is ESP. Example: -LNG ENG,ESP	Not required. If this parameter is not given, then the languages that are already set in the project will be used as the default.
------	--	---

Example

```
PSTOOLS.EXE -CT MICROSOFT -CD CJR750 -CO PTMDO -CP PTMDO -PJC CJR1 -TD CJR7502 -
TO PTMDO -TP PTMDO -LF C:\TEMP\CJR1.LOG -CL 150 -AF 0 -DDL 0 -OBJ 0,1,5,10,20 -
RST 0 -LNG ENG, ESP
```

Object Type Selection Table

Number	Object Description
0	Record
1	Index
2	Field
3	Format Definition
4	Translate
5	Page
6	Menu
7	Component
8	Record People Code
9	Menu People Code
10	Query
11	Tree Structure

12	Tree
13	Access Group
14	Color
15	Style
16	<i>Not used</i>
17	Business Process
18	Activity
19	Role
20	Process Definition
21	Process Server
22	Process Type
23	Process Job
24	Process Recurrence
25	Message
26	Dimension
27	Analysis Model
28	Cube Template
29	Interface Object
30	File layout Definition
31	Component Interface
32	App Engine Program
33	App Engine Section
34	SQL Object
35	Message Node
36	Message Channel
37	Message Definition
38	Approval rule set
39	Message People Code
40	Subscription People Code
41	Message Channel People Code
42	Component Interface People Code
43	Application Engine People Code
44	Page People Code

45	Page Field People Code
46	Component People Code
47	Component Record PeopleCode
48	Component Record Field PeopleCode
49	Image
50	Style Sheet
51	HTML Catalog
52	File Reference objects

CHAPTER 7

EDI Manager

The EDI Manager provides the tools you need to manage electronic commerce transactions with your trading partners. You use it to set up and maintain data about your trading partners, and to define the data mapping that occurs between transaction files and the tables in your PeopleSoft database. The following table lists tasks you may have to perform and the corresponding documentation sections.

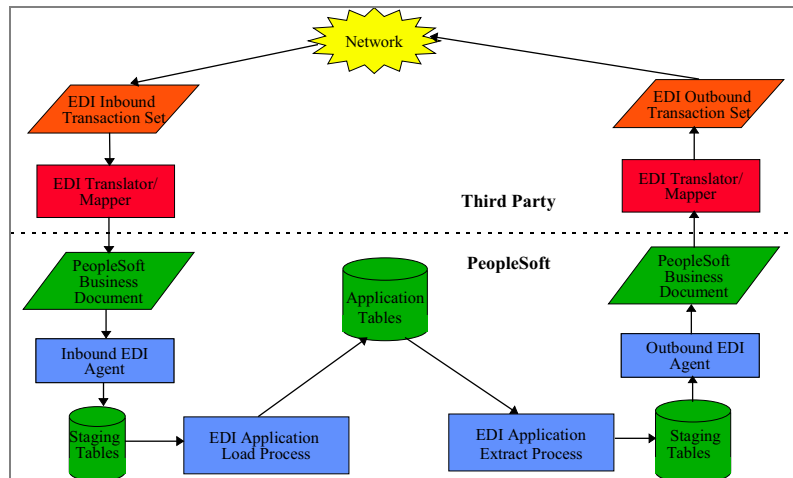
<i>If you need to...</i>	<i>Read this section...</i>
Learn about electronic commerce architecture	Understanding Electronic Data Interchange
Specify conversion rules for transaction data	Converting EDI Codes and PeopleSoft Codes
Define the transaction types your application can process	Defining EDI Transactions
Set up trading partner relationships between external companies and internal business units	Setting Up Trading Partners
Map data between EDI transaction files and staging tables in your PeopleSoft database	Mapping EDI Transactions
Perform ongoing administration tasks involved in keeping EDI processing running smoothly	Monitoring EDI Processing

Understanding Electronic Data Interchange

Electronic Data Interchange (EDI) is a standard means of exchanging data between companies, so they can transact business electronically. For example, using EDI, a company can submit an order to a vendor, and the vendor can acknowledge and fulfill the order, without any paper changing hands or any contact between company representatives.

EDI provides a standard format for transaction data, allowing trading partners to communicate in a common language. When one company needs to initiate a transaction with another, it extracts the transaction data from its database, translates it into the common EDI format, and transmits it over a network to the trading partner. The second company receives the EDI transmission and transfers its data into its transaction processing application.

This conceptually simple process involves several distinct steps. Here's an overview of the architecture that enables PeopleSoft application to complete EDI transactions:



PeopleSoft EDI Architecture

The portion above the dotted line shows the services typically provided by a Value-added Network (VAN). A VAN is a consulting organization that provides services related to the exchange of EDI transactions—usually a private network for exchanging EDI transactions, although the network could also be the Internet.

For incoming transactions, the hand-off to PeopleSoft applications comes when the EDI translation software converts transactions from the standard EDI formats X.12 or EDIFACT into PeopleSoft Business Document format. A PeopleSoft-supplied EDI Agent reads the PeopleSoft Business Document files and places their data in staging tables in the PeopleSoft database. An EDI Application Load SQR transfers data from the staging tables to the appropriate application tables. From there, the data is processed like any other PeopleSoft transaction.

The process for outgoing transactions is similar. You periodically run an SQR that extracts the appropriate data from the application tables and copies it to a set of staging tables. An EDI Agent creates PeopleSoft Business Documents from this data; then the EDI translation software converts the documents into X.12 or EDIFACT format and transmits them to the trading partner.

So, the PeopleSoft portion of the process has two stages:

- Based on data mappings you define, EDI Agents copy data between PeopleSoft Business Documents and the incoming or outgoing staging tables.
- SQRs transfer data between the staging tables and the application tables, performing any necessary data validation.

You use the EDI Manager to define the mappings that EDI Agents use for conversions and to maintain information about your vendors and other trading partners.

The staging tables are a temporary storage area for EDI transaction data. Later, a transaction-specific “application load” process transfers the data from the staging tables into the application tables.



For more information about the application load and application extract processes, see the documentation for the application that supports the transactions.

Converting EDI Codes and PeopleSoft Codes

When you receive an EDI transaction from a trading partner, much of its data is in the form of codes or identification numbers. For example, the trading partner who submitted the transaction is given as a Trading Partner ID, and the type of transaction is identified by a Transaction ID.

Your PeopleSoft database also stores much of its data in the form of codes or IDs. You have Business Unit IDs, Customer IDs, Employee IDs, Operator IDs, and so on. Some of these codes and IDs represent the same data as the codes and IDs in the EDI transaction, even though the codes themselves might be different.

When the EDI Agent processes a transaction, it copies the transaction data into the PeopleSoft database. As it does so, it can convert the external EDI codes into the internal PeopleSoft codes. Similarly, as it writes out the file for an outgoing EDI transaction, it can convert the PeopleSoft codes into codes that your trading partner will recognize. You specify how the EDI Agent converts the codes using the EDI Manager.

The EDI Manager can perform two kinds of conversions:

- It can translate between EDI event codes and PeopleSoft action codes, which specify what action a transaction requires. See [Action Codes and Event Codes](#).
- It can convert data values from any field in the transaction. See [Data Values](#).

Action Codes and Event Codes

EDI transactions use *event codes* to specify what action the transaction calls for. For example, event codes specify whether the current transaction is a completely new transaction, a resubmitted transaction, or an update to a previous transaction.

Event codes come in two flavors, both of which can be specified for the same transaction:

- *Primary event codes*, also called *purpose codes*, specify the status of the transaction: whether it's a new transaction, a cancellation, a duplicate, a status request, and so on. Every transaction has a primary event code assigned to it.
- *Secondary event codes*, also called *transaction codes*, specify the type of transaction in detail. For example, a transaction's secondary event code could say that the transaction is a catalog order, a rush order, or a request for a sample. Not all transaction types include secondary event codes.

PeopleSoft applications, on the other hand, determine what action to take on a transaction using a single *action code*. So, when the EDI Agent processes an EDI transaction, it needs to convert its event code(s) into a PeopleSoft action code.

Using the EDI Manager, you specify which (pairs of) event codes get translated into which action codes. Since you may want to process transactions differently depending on which trading partner they come from, the EDI Manager enables you to define different event code-to-action code translations for each trading partner.

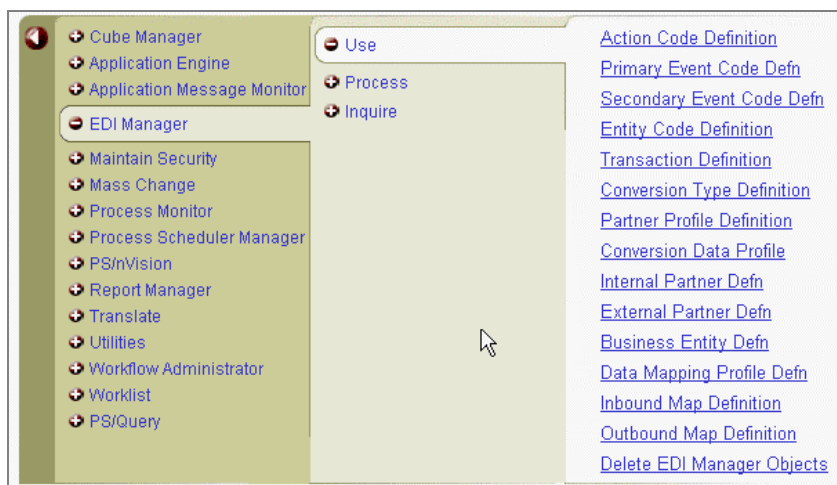
The following table lists the three major steps in defining how the EDI Agent translates event codes into action codes and where to find information.

Steps for defining how to translate codes	Use this section...
Define the set of recognized primary event codes, secondary event codes, and action codes.	Link to next section
As part of a trading partner profile, specify which event codes or pairs of event codes correspond to which action codes.	Defining EDI Transactions
For each of your trading partners, specify which trading partner profile to use.	Setting Up Trading Partners

Defining Primary and Secondary Event and Action Codes

To access EDI Manager from your browser

1. Open your browser. Select **PeopleTools, EDI Manager, Use**.



EDI Manager Use Options

2. Select an option from the list.

The following sections describe the Use options.

To define a primary event code

1. Select Primary Event Code Defn.

For each of the options, you can search for an existing value or directly add a new value by clicking on the link at

Find an Existing Value – Primary Event Code

Add a New Value – Primary Event Code

2. Click **Add a New Value** then add the value.

Enter the primary event code as it will appear in the PeopleSoft Business Document

3. Enter a description of the event code and save the page.

This Description is for your information only. It doesn't affect the processing of a PeopleSoft Business Document.

To define a secondary event code

1. Select Secondary Event Code Defn.

2. Enter the secondary event code as it will appear in the PeopleSoft Business Document.

Alternatively, you can search for an existing Secondary Event Code, by clicking Search.

3. Enter a description of the event code and save the page.

This Description is for your information only. It doesn't affect the processing of a PeopleSoft Business Document.

To define an action code



We deliver the EDI Manager with a complete set of action codes for the actions standard PeopleSoft applications support. You need to add new action codes only if you create new programs that handle new actions.

1. Select Action Code Definition.
2. Enter the action code as you want the EDI Agent to write it in the electronic commerce staging tables.

Alternatively, you can search for an existing Action Code, by clicking Search.

3. Enter a description of the action code and save the page.

This Description is for your information only. It doesn't affect processing.

Data Values

In many cases, the set of possible values in a particular field of an EDI transaction doesn't match the corresponding set of values in the PeopleSoft database. For example, the EDI transaction might identify bank transaction codes using three-digit numbers, while the PeopleSoft database uses single letters to represent the same codes. In such cases, the EDI Agent needs to translate the external values into the corresponding internal values.

To accomplish this task, you need to define a *conversion data profile*. A conversion data profile takes the values from a particular PeopleSoft database table (such as the table holding bank transaction codes) and specifies how that value appears in PeopleSoft Business Documents. You can create multiple conversion data profiles for the same table, because you might need to create different conversions for different trading partners. For example, different banks might use different transaction codes.

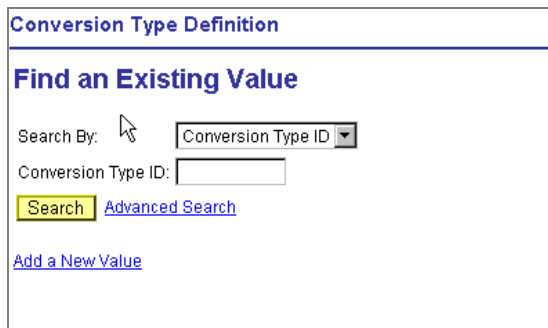
Here are the three major steps in specifying how to convert the data values for your trading partners.

Steps for converting data values	Use this section...
Identify a table in the PeopleSoft database whose values appear differently in PeopleSoft Business Documents. Assign a Trading Partner Conversion ID to the table.	Identifying Database Table for Conversion
Create a conversion data profile that specifies the internal and corresponding external values for each value in the table. If different trading partners use different values, create one conversion data profiles for each set of external values.	Creating a Conversion Data Profile
Assign the appropriate conversion data profile to each of your trading partners.	Setting Up Trading Partners

Identifying Database Table for Conversion

To identify a database table whose values need to be converted

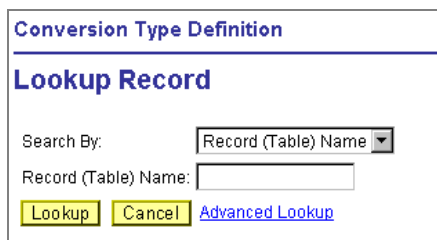
1. Select Conversion Type Definition.
2. Enter a Conversion Type ID or search for an existing one.



The screenshot shows a web interface titled "Conversion Type Definition". Below the title is a section header "Find an Existing Value". Under this header, there is a "Search By:" label with a mouse cursor pointing to a dropdown menu that currently displays "Conversion Type ID". Below the dropdown is a text input field labeled "Conversion Type ID:". At the bottom of the search section, there are two buttons: a yellow "Search" button and a blue "Advanced Search" link. Below the search section is a blue link labeled "Add a New Value".

Conversion Type Definition ID

You can also search for the record name.

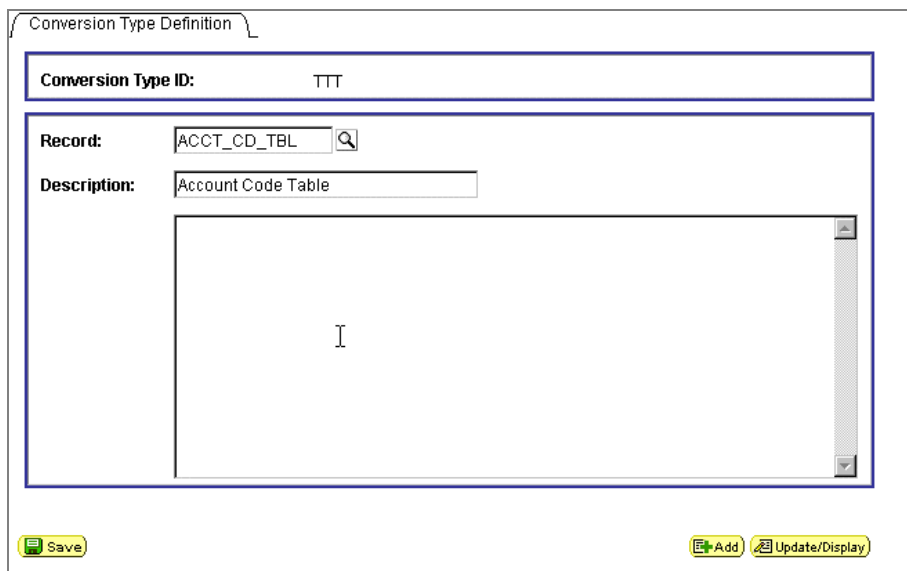


The screenshot shows the same "Conversion Type Definition" window, but with the "Lookup Record" section selected. Under this header, there is a "Search By:" label with a dropdown menu that currently displays "Record (Table) Name". Below the dropdown is a text input field labeled "Record (Table) Name:". At the bottom of the lookup section, there are three buttons: a yellow "Lookup" button, a yellow "Cancel" button, and a blue "Advanced Lookup" link.

Lookup Record Name Page

3. Enter the record name of the table whose values you want to convert and enter a description.

You can enter a description in the Description text box, the large edit box, or both.



Conversion Type Definition

Conversion Type ID: TTT

Record: ACCT_CD_TBL

Description: Account Code Table

Save Add Update/Display

Conversion Type Definition Page

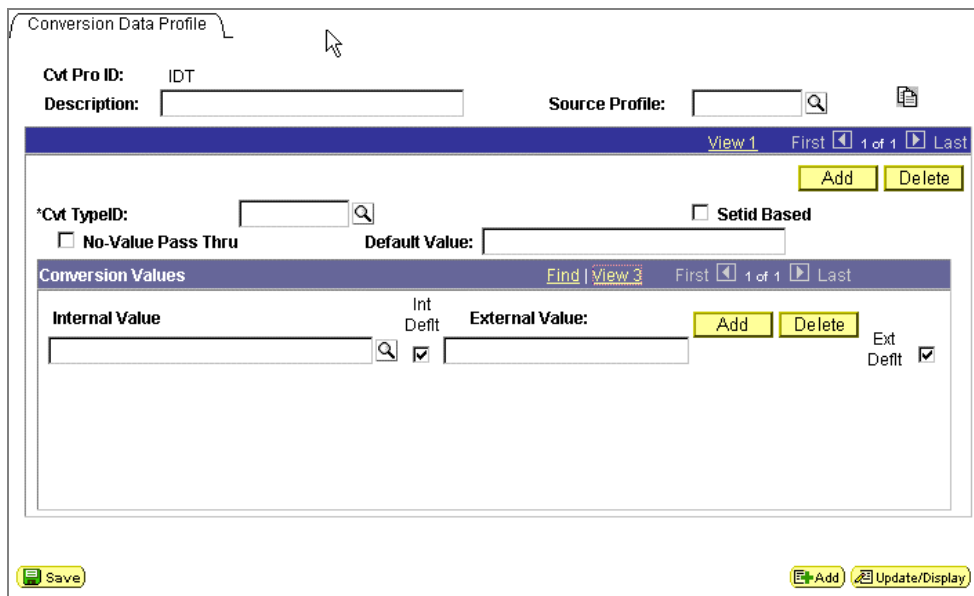
4. Save the page.

Creating a Conversion Data Profile

To create a conversion data profile

1. Select Conversion Data Profile.

Search for an existing Conversion Data Profile or enter a new value.



Conversion Data Profile

Cvt Pro ID: IDT

Description:

Source Profile:

View 1 First 1 of 1 Last

Add Delete

*Cvt TypeID:

☐ No-Value Pass Thru ☐ Setid Based

Default Value:

Conversion Values Find View 3 First 1 of 1 Last

Internal Value	Int Deflt	External Value:	Ext Deflt
	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>

Save Add Update/Display

Conversion Data Profile page

You use this page to specify translation values for one or more database tables, identified here by their Conversion Profile IDs.

The page includes two areas to enter data. The outer area enables you to add multiple Conversion Type IDs to the conversion profile. The inner area scrolls among the values in the table identified by the Conversion ID.

2. Copy the definition of a similar conversion data profile (optional).

If the profile you're creating is similar to an existing profile, you can copy the details of the existing profile definition into this component. Then, you can change just the parts that are different.



Select the existing profile in the **Source Profile** list box, then click the button. The system copies the translation values from the selected profile into the pages.

3. Enter a description of the conversion data profile.

You can enter both a short description (in the **Description** text box) and a long description (in the long edit box). These descriptions are for your information only; the system doesn't use them.

4. Select a Conversion ID.

In the **Cvt Pro ID** drop-down list box, select the Trading Partner Conversion ID of the table that you want to provide external values for.

5. Specify whether the table whose values you are converting has a Set ID key.

Many tables in PeopleSoft Financials applications have a Set ID key. The Set ID key enables you to store data for multiple business units on the same table while making sure that each unit only accesses its own data. The Conversion Data Profile page enables you to select a Set ID. Selecting a Set ID will control the list of internal values available for mapping: if the table you are retrieving the internal values from has a Set ID field, the Conversion Data Profile page will only display the values for the selected Set ID.

To select a Set ID, click the **Setid Based** check box, then select the appropriate Set ID from the drop-down list next to the check box.

6. Select an internal value and enter the corresponding external value.

In the **Internal Value** list box, select a value from the table whose Conversion ID you selected. Then, in the **External Value** text box, type the corresponding value as it will appear in PeopleSoft Business Documents.

7. Specify which internal and external values to use when there are multiple possibilities.

One potential complication of having many-to-one mapping occurs with outbound transactions: when the internal value is D, what value should the Outbound EDI Agent write into the outbound file? In this situation, the Outbound EDI Agent uses the row marked as the internal default (**Int Deflt**) value).

The same scenario holds in reverse when you have multiple internal database values that you need to map to the same external value. There is no problem with outbound transactions: the Outbound EDI Agent maps the internal value to the appropriate external value. For incoming transactions, however, you need to specify which of the multiple internal values to use. You do so by selecting one of the rows as the external default (**Ext Deflt**) value.

When you need to convert multiple external values to the same internal value, you need to mark *each* of the external values as an **Ext Deflt**, then mark *one* of the rows as the **Int Deflt**. When you need to convert multiple internal values to the same external value, mark each of the internal values as an **Int Deflt**, then mark one row as the **Ext Deflt**.



You need to perform this step even if you have all one-to-one mappings. Simply select the **Int Deflt** and **Ext Deflt** check boxes for every row.

8. Repeat the previous steps to add additional Conversion IDs. Then save the page.

Defining EDI Transactions

When you receive an EDI transaction from a trading partner, the first record of the transaction includes a *Transaction ID* that identifies the transaction type. The EDI Agent uses the Transaction ID, in conjunction with the Trading Partner ID, to determine which inbound map to use to process the transaction data. Similarly, when you initiate an outbound transaction, the EDI Agent puts the appropriate Transaction ID in the first transaction record so that the recipient knows what kind of transaction you've sent.

In the EDI Manager, you need to identify what transactions your system is prepared to process. You also need to specify, for each of your trading partners, which of these transactions they are authorized to perform. Here are the major steps you need to take:

Steps for converting data values	Use this section...
Create a Transaction ID for each type of transaction your applications support.	Defining EDI Transactions
Create one or more <i>partner profiles</i> , which list a set of transactions that partners can perform.	To define a partner profile
Assign a partner profile to each trading partner.	Setting Up Trading Partners

Defining Transactions

To define a transaction

1. Select Transaction Definition.
2. Enter a unique Transaction ID as it will appear in PeopleSoft Business Documents.

- Specify whether you're setting up an ID for inbound or outbound transactions.

Transaction Definition

Add a New Value

EC Transaction ID:

Inbound / Outbound Switch:

[Find an Existing Value](#)

Specifying Inbound or Outbound Transaction Definition



If you support both inbound and outbound transactions of the same transaction type, you have to define two Transaction IDs.

- Click the **Add** button.

The Transaction Definition page displays.

Transaction Definition

Trans ID: TNT Inbound

Descr:

Transaction Option Definition Find | View All First 1 of 1 Last

EC Option	Description
<input type="text"/>	<input type="text"/>

Transaction Option Values Find | View All First 1 of 1 Last

Op Value	Description
<input type="text"/>	<input type="text"/>

Transaction Definition page

You use this page to define the transaction and specify any special transaction parameters.

- Enter a description of the transaction in the **Descr** text box.
- The description appears in list boxes when you need to select a Transaction ID.
- Enter the name of a transaction option that the application load or application extract process

uses.

The **EC Option** text box gives you a place to specify a transaction parameter that the system will use when it copies data between the electronic commerce staging tables into the transactional application tables.

Enter the name of the option in the **EC Option** text box and a description in the **Description** text box.

7. Enter the valid values for the transaction option.

In the **Transaction Option Values** area, you need to list the valid values for the transaction option you specified at step 6. When you define a trading partner, you'll specify which of these values to use for that partner. In this way, the application load or application extract process can handle the transaction differently depending on the trading partner it comes from.

Enter a value in the **Op Value** box, and describe this value in the **Description** text box.

Repeat steps 6 and 7 to add additional transaction options.

8. Save the page.

To define a partner profile

1. Select Partner Profile Definition.

To define a new partner profile, enter a unique EC Profile ID in the dialog box, and click Add. Otherwise, search for an existing value for the desired EC Profile ID.

The screenshot shows a web-based form titled "Partner Profile Definition Page". At the top, there are two tabs: "Profile Definition" (selected) and "Profile Defaults". The form contains the following elements:

- EC Profile ID:** A text box containing the value "PCC".
- Source TPID:** A text box with a search icon and a small document icon.
- Description:** A text box.
- EC Outbound File List Path:** A text box.
- EC Outbound File List Name:** A text box.
- New List File Per Run:** A checkbox that is currently unchecked.
- Message:** A yellow box with the word "Message" inside.
- Buttons:** "Save" (green), "Previous tab" (grey), and "Next tab" (yellow).
- Navigation:** A link "Profile Definition | Profile Defaults" at the bottom.


Partner Profile Definition Page

You use this page to define the profile and specify where files for this partner reside.

2. Copy the definition of a similar partner profile (optional).

If the profile you're creating is similar to an existing profile, you can copy the details of the existing profile definition into this component. Then, you can change just the parts that are different.



Select the existing profile in the **Source TPID** list, then click the  button. The system copies the translation values from the selected profile into the pages.

3. Enter a description of the profile in the **Description** text box.
4. Specify the directory where you want the EDI Agent to write outbound transaction files for partners with this profile.

Enter the directory path in the **EC Outbound File List Path** text box.

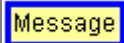


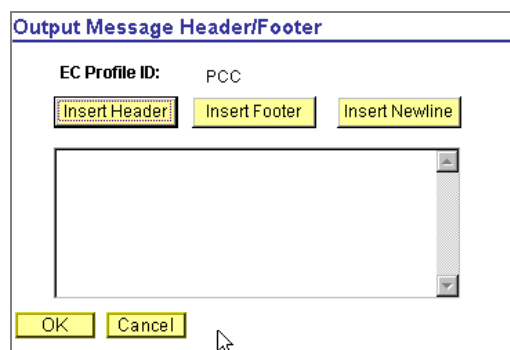
If the EDI Agent runs on a server running MVS, it ignores this setting. Instead, it looks in the JCL for a DDNAME matching the filename you specify in the next step.

5. Enter a name for the file that lists all pending outbound transaction files.

As the EDI Agent creates transaction files, it adds the names of those files to a list file, whose name you enter in the **EC Outbound File List Name** text box.

The transaction files themselves will be named 1.DAT, 2.DAT, and so on, except on MVS servers. On MVS, the files will be named using the first eight characters of the external Trading Partner ID.

6. Check the **New List File Per Run** box if you want a new list generated every run. (optional)
7. Click  to designate message header and footer. The message page appears.



The dialog box titled "Output Message Header/Footer" contains the following elements:

- EC Profile ID:** PCC
- Three buttons: **Insert Header**, **Insert Footer**, and **Insert Newline**.
- A large text area for entering the message content.
- At the bottom, **OK** and **Cancel** buttons.

8. Select the Profile Defaults tab.

Profile Defaults Page

Use this page to specify which transactions a partner with this profile can perform and how the EDI Agent converts event codes into action codes for each transaction. If a transaction has transaction options associated with it, you also specify which value to assign to that option for partners with this profile.



For more information about action codes and event codes, see *Converting EDI Codes and PeopleSoft Codes*. For information about transaction options, see the procedure *Defining Transactions*.

9. Select a transaction type that partners with this profile can perform.

Select the Transaction ID in the EC Transaction ID, and specify whether it's an inbound or outbound transaction by selecting I or O from the In/Out list.

10. If the transaction type has transaction options associated with it, specify which value to use for each one.

When you define a transaction type, you have the option to specify one or more transaction options that the system will use when it copies data between the electronic commerce staging tables into the transactional application tables. If the transaction type you're adding includes options, you need to specify which of the available values for that option the EDI Agent should use for partners with this profile.



Most transaction types will not have any options, and you can skip this step.

Select the option from the EC Option list. Then, select one of its valid values from the Op Value list. Repeat this step for each defined transaction option.

11. Specify which action code to use for each pair of primary and secondary event codes.

The controls in the Action Assignment group box enable you to specify how the EDI Agent translates between the event codes in a PeopleSoft Business Document and the action codes in the electronic commerce staging tables.

Select one of the defined action codes in the EC Action Code box, then select the primary and secondary event codes that map to this action code. If this transaction type only uses a primary event code (**Pri Event**), you can leave the Sec Event box empty.

Repeat this step to define each possible action code, and each possible combination of primary and secondary event codes, for this transaction type. Press F7 to add additional rows to the scroll.

12. Repeat steps 7 to 9 for each transaction type that partners with this profile can perform.

13. Save the component.

Setting Up Trading Partners

In casual speech, the term trading partner refers to a company with which your company does business. That's close to what it means in the EDI Manager too, except that it needs to be more specific. In EDI Manager terms, a trading partner is a business entity in an external company with which a business entity in your company does business.

PeopleSoft Financials applications enable you to organize your company into multiple business units, each of which tracks its data somewhat independently. Each business unit likely does business with a different set of trading partners. When a trading partner submits an EDI transaction, it needs to address it to a specific business unit, or the EDI Agent needs to know in some way which business unit to forward the transaction to.

Conversely, the external companies you do business with are not monolithic entities. They have different divisions, departments, or business units, and you may need to address your EDI transactions to a specific entity inside that company.

When you set up trading partners in the EDI Manager, you need to define the internal structure of your company and possibly of the companies you trade with also. You should:

- Define Entity Codes for the various types of business entities that serve as trading partners. Internally, the entities are usually business units.
- Assign Trading Partner IDs to the internal entities (business units) that external trading partners need to be able to submit transactions to. Because of the way each PeopleSoft Financials application tracks its own set of business units, a single Trading Partner ID can refer to multiple Business Unit IDs, all of which actually refer to the same physical business unit.
- Assign Trading Partner IDs to the external companies with which you do business.
- If you need to submit EDI transactions to specific business entities within an external company, assign a Business Entity ID to each one.

To define a PeopleSoft Entity Code

1. Select Entity Code Definition.

To define a new entity code, select **Add a New Value**, enter a unique Entity Code in the dialog box, and click Add.

To update an existing entity code, select **Search** or **Find an Existing Value** to select from a list of available codes.

Ec Entity Code Tbl

PeopleSoft Entity Code: APBU

Description: Accounts Payable Business Unit

Record (Table) Name: BUS_UNIT_TBL_API ☐ External Entity

Save Return to Search Add Update/Display

Entity Code Definition page

Use this page to specify what type of business entity this Entity Code applies to, and which table in the PeopleSoft database lists the valid entities of this type.

2. In the **Description** text box, describe to what sort of entity this Entity Code applies.
3. Select the record definition for the table that lists valid entities of this type.

In other words, pick the prompt table for this type of entity from the Record (Table) Name list. For example, if you're defining an Entity Code for business units, specify the table that holds Business Unit IDs.

4. Specify whether this Entity Code applies to external trading partners.

You need to identify Entity Codes both for your internal business entities and for the external companies you trade with. For example, you'll probably want to create Entity Codes based on the Vendor table and the Customer table.

If the Entity Code you're defining is for external companies, select the **External Entity** check box. If the Entity Code is for entities internal to your own company, leave the check box unselected.

5. Save the page.

To create an internal Partner ID

1. Select Internal Partner Defn.

To define a new definition, select **Add a New Value**, enter a unique Internal Partner ID in the dialog box, and click Add.

To update an existing definition, select **Search** or **Find an Existing Value** to select from a list of available values.

The screenshot shows the 'Ec Int Partner Def' form. It contains the following fields and controls:

- Int TPID:** A text box with the value 'GENERALINT'.
- Descr:** A text box with the value 'General Internal Partner'.
- Business Unit Assignment:** A section with a table containing two rows:

*PS Code	*Unit
GENR	GENR

 To the right of the table are 'Add' and 'Delete' buttons. Above the table are links for 'Find | View All' and 'First 1 of 1 Last'.
- TP ID Alias Definition:** A section with two text boxes: '*Ext TPID' and '*Alias TPID'. To the right are 'Add' and 'Delete' buttons. Above the boxes are links for 'Find | View All' and 'First 1 of 1 Last'.
- Buttons:** At the bottom of the form are buttons for 'Save', 'Return to Search', 'Add', 'Update/Display', and 'Correct History'.

Internal Partner Definition page

Use this page to specify which internal business entities share this Partner ID, and which external trading partners do business with this partner.

2. Enter a description of the Partner ID in the **Descr** text box.
3. Specify to which business entity or entities this Trading Partner ID applies.

In the **PS Code** box, select the Entity Code for the type of business entity, then select the specific entity in the Unit box.

Since the same business unit can be referred to by different Business Unit IDs in different PeopleSoft Financials applications, you can add multiple units to the same Trading Partner ID.

4. Specify which external trading partners work with this internal trading partner, and what name they use to refer to the internal partner.

In the **Ext TPID** box, select the name of an external trading partner that does business with the internal partner you're defining. Then, in the **Alias TPID** box, enter the name by which the selected external partner will refer to the internal partner—that is, the text that will appear in the Internal Trading Partner field in the PeopleSoft Business Document.



If you haven't defined your external trading partners yet, you'll need to return to this page after you add them.

Repeat this step for each external trading partner.

5. Save the page.

To add an external trading partner

1. Select External Partner Defn.

To define a new definition, select **Add a New Value**, enter a unique External Partner ID in the dialog box, and click Add.

To update an existing definition, select **Search** or **Find an Existing Value** to select from a list of available values.

You'll need to find out from the trading partner or from the VAN what Trading Partner ID to use.

Ec Ext Partner Def

Ext TPID: TNT

Trading Partner Definition

Descr:

Map ID: Profile ID: Cvt Pro ID:

Customer / Vendor Assignment Find | View All First Last

*PS Code	SetID	*PS Customer/Vendor Number	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="Add"/> <input type="button" value="Delete"/>

TP Alias Assignment Find | View All First Last

*Int TPID	*Alias TPID	
<input type="text"/>	<input type="text"/>	<input type="button" value="Add"/> <input type="button" value="Delete"/>

External Partner Definition page

Use this page to provide information about the external trading partner.

2. Enter a description of the trading partner in the **Descr** text box.
3. Select the map profile that includes the mappings this trading partner uses.

Select the appropriate Map Profile ID from the list of valid Map IDs.

4. Select the partner profile that identifies the transactions this partner can perform.

Select the appropriate Partner Profile ID from the list of valid Profile IDs.

5. Select the conversion data profile that specifies how to translate this partner's codes into PeopleSoft codes (optional).

Select the appropriate Conversion Data Profile ID from the **Cvt Pro ID** list.

6. Specify the Customer ID, Vendor ID, or other ID for this trading partner.



If you exchange transactions with multiple business entities inside this trading partner, you'll need to define the partner's external business entities. See the next procedure, To define business entities in an external trading partner company. If you define external business entities, each one needs to have its own unique Customer ID, Vendor ID, or other ID. The ID you assign here should be the ID for the corporate office.

In the **PS Code** box, select the Entity Code for the type of entity this partner is—customer, vendor, or whatever. If your system uses table set processing, select the **SetID** for the Customer or Vendor table in the **SetID** box. In the **PS Customer/Vendor Number** box, select the specific value from the Customer or Vendor table that corresponds to this trading partner.

If different tables in the system refer to this trading partner by different names, you can list all the IDs as part of the trading partner definition.

7. Identify the internal trading partners who do business with this external trading partner.

In the **Int TPID** box, select the name of an internal trading partner that does business with the external partner you're defining. Then, in the **Alias TPID** box, enter the name by which the new external partner will refer to the selected internal partner—that is, the text that will appear in the Internal Trading Partner field in the PeopleSoft Business Document.

Repeat this step for each internal trading partner.

8. Save the page.

To define business entities in an external trading partner company



Follow this procedure only if you need to be able to address transactions to multiple business entities that are part of the same external trading partner.

1. Select Business Entity Defn.

To define a new definition, select **Add a New Value**, enter a unique External Partner ID in the dialog box, and click Add.

To update an existing definition, select **Search** or **Find an Existing Value** to select from a list of available values.

Business Entity Definition Page

Use this page to define the groups or entities you deal with inside an external trading partner.

2. Enter a description of the business entity in the **Descr** text box.
3. Specify to which trading partner company this business entity is a part.

Select the appropriate Trading Partner ID from the Parent Trading Partner ID list.

When the EDI Agent processes a transaction that involves the business entity you're defining, it will use the Map ID, Profile ID, and Conversion Data ID assigned to the parent trading partner.

4. Specify the Customer ID or Vendor ID for this business entity.

To establish a business entity as a valid trading partner, it must appear as a separate entry in your PeopleSoft database as a customer or a vendor. In the **Customer/Vendor Assignment** area, you identify which customer or vendor will be referred to by the External Business Entity ID you're creating. The system needs this information so that it can figure out what Customer ID or Vendor ID to assign to incoming EDI transactions, and what External Business Entity ID to assign to outgoing EDI transactions.

In the **PS Code** box, select the **Entity Code** for customers or vendors. If your system uses table set processing, select the SetID for the Customer or Vendor table in the SetID box. In the PS Customer/Vendor Number box, select the specific value from the Customer or Vendor table that corresponds to this business entity.

If different tables in the system refer to this business entity by different names, you can list all the IDs as part of the trading partner definition.

5. Identify the internal trading partners who do business with this external trading partner, then save the page.

In the **Internal Entity ID** box, select the name of an internal trading partner that does business with the external entity you're defining. Then, in the **Alias Internal Entity ID** box, enter the name by which the new external entity will refer to the selected internal partner—that is, the text that will appear in the Internal Trading Partner field in the PeopleSoft Business Document.

Repeat this step for each internal trading partner.

Deleting EDI Manager Objects

There may come a time when you need to remove business objects that you have created. You may wish to delete the business entities that have been established if the business entities are no longer valid. This is done through the **Delete EDI Manager Objects** menu list.

To delete EDI Manager objects

1. Select Delete EDI Manger Objects.

Ec Delete

Objects To Be Deleted

Delete EC Map:

Delete Trading Partner Profile:

Delete Data Conversion Profile:

Delete External TP ID:

Delete Internal TP ID:

Press SAVE To Commit Delete

Save

Delete EDI Manager Objects page

2. Enter the objects to be deleted in the appropriate field list.
3. Press save to delete.

CHAPTER 8

Mapping EDI Transactions

PeopleSoft applications store their transaction data in tables in the PeopleSoft database. For an application to process an EDI transaction, it needs the transaction data in its tables. So, the first step in processing an EDI transaction is transferring the data from an incoming EDI Transaction Set file into the application tables. Similarly, the first step in generating an EDI transaction for delivery to a trading partner is getting the transaction data out of the application tables into an EDI Transaction Set file.

You specify how the EDI Agent transfers data between EDI Transaction Set files and application tables by creating *electronic commerce maps*. There are two kinds of electronic commerce maps:

- Inbound maps, which transfer incoming transaction data into your PeopleSoft database
- Outbound maps, which create outgoing EDI documents from transaction data in the database

This chapter explains how to create both types of maps. It also tells you how to create *map profiles*, which specify to what maps your trading partners have access.

Processing New EDI Transactions

We deliver PeopleSoft applications with a number of EDI transactions already defined. However, if you want your application to process a transaction we don't supply, you can use PeopleTools to add support for it. Here are the major steps in the process.

To process new EDI transactions

1. Use PeopleTools to develop the application that processes the transaction. Create the database tables and the pages users need to process the transaction.
2. Define a PeopleSoft Business Document format based on the EDI format for the transaction. At this step, think about how the data from the EDI document needs to map to the application tables.
3. Write a translation program that converts EDI documents into PeopleSoft Document format. Usually, you'll have an EDI Translator program to handle this step.
4. Create staging tables to serve as a temporary holding area between the PeopleSoft Document and the application tables.
5. Create a mapping that copies data between the PeopleSoft Business Documents and the staging tables.

6. Write an application load procedure that transfers data from the staging tables to the application tables.

This chapter primarily covers step 5 in this procedure.

PeopleSoft Business Document Format

A PeopleSoft Business Document is a ASCII file that contains the data for one or more EDI transactions. It contains both the transaction data and control information, which tells the system what type of transaction it is, who it came from, and how the various parts of the document relate to each other.

A PeopleSoft Business Document is divided into *records*, separated from each other with a line feed and carriage return. Each record is divided into fixed-length fields. One of the fields in the record, usually the first one, is a Record ID field that specifies what type of record it is—a header, detail line, summary line, or whatever. What the rest of the fields are depends on the type of record.

The first record in a PeopleSoft Business Document is always a *control record*. The control record specifies the type of transaction contained in the following lines and the trading partners involved. Based on the control record, the EDI Agent retrieves the mapping definition for the specified transaction and the data conversion options relevant to the trading partner.

The EDI Agent reads the records in the document one at a time. For each record following the control record, the EDI Agent reads the value in the Record ID field and uses it to determine which PeopleSoft record definition to use to parse the rest of the record. The PeopleSoft record definition specifies the location, size, and data type of the remaining fields in the record. The EDI Agent copies the data from the document into staging tables in the database, following the rules in the mapping definition.

When the EDI Agent encounters a new control record—identified by the Record ID 999 or 998—it retrieves the mapping definition for the new transaction type and repeats the process.

Most transactions include a variety of record types: a header, detail lines, schedules, summary lines, and so on. The layout of the records within a PeopleSoft Business Document follows a logical pattern. All detail lines linked to a particular parent or header line must follow the parent record.



The EDI Agent processes transactions at the “unit of work” level, corresponding to the ST/SE level in X.12 format. Each transaction should be a unit of work that you want the EDI Agent to commit or rollback as a unit, such as a single purchase order or invoice.

Let’s look at an example.

```
001 Header Line 1
002 Detail Line 1A
002 Detail Line 1B
002 Detail Line 1C
001 Header Line 2
```

002 Detail Line 2A
002 Detail Line 2B

This file has two header lines and five detail lines. The EDI Agent determines which detail lines go with which header lines based entirely on their order.

When the EDI Agent processes this PeopleSoft Business Document, it works through the records in order. First, it processes Header Line 1 using the work record that's appropriate for Record ID 001. Next, it processes Detail Line 1A using the work record for Record ID 002.

Now, if you ask the EDI Agent to calculate some value, such as the sum of all the detail lines, it determined which detail lines to group together based on their position. It calculates the value for all detail lines *since the most recently encountered header line*.

Control Records

The first record in a PeopleSoft Business Document is always a control record (998 and 999 records). The control record specifies the type of transaction contained in the following lines and the trading partners involved. Based on the control record, the EDI Agent retrieves the mapping definition for the specified transaction and the data conversion options relevant to the trading partner.

Within the flat file, the control records:

- For Outbound — Allow a third party translator or trading partner to determine what the data contains (999 only).
- For Inbound — Allow the EDI Manager to determine what the data contains (999 and 998).

'999' Record Format

This reserved record identifier acts as a control record to “switch” map definitions within a data file. The layout is as follows :

Field	Description	Value
1-3	Record Identifier	'999'
4-18	Transaction ID	Char
19-22	External Entity Code	Char / Values : CUST or VNDR
23-38	External Trading Partner ID	Char
39-42	Internal Alias Entity Code	Char / Values : AP, AR, OM, ...
43-58	Internal Trading Partner Alias ID	Char

‘998’ Record Format

This format allows the specification of a PeopleSoft Map ID directly in the data contents. This type of map definition can be used when Trading Partner conversions are not needed or not available. All other data conversion functionality can be utilized.

<i>Field</i>	<i>Description</i>	<i>Value</i>
1-3	Record Identifier	‘998’
4-13	PeopleSoft EC Map ID	Char
14-23	PeopleSoft Trading Partner Profile ID	Char
24-33	PeopleSoft Data Conversion Profile ID	Char

Creating Electronic Commerce Maps

Electronic commerce maps specify how the EDI Agent transfers data between PeopleSoft Business Documents and the staging tables in the PeopleSoft database. There are two types of maps:

- Inbound maps, which the EDI Agent uses to transfer data from PeopleSoft Business Documents to the staging tables, in preparation for processing by a PeopleSoft application.
- Outbound maps, which the EDI Agent uses to create PeopleSoft Business Documents from data the applications put in the staging tables for delivery to a trading partner.

You create one map for each EDI transaction.



Electronic commerce maps help transfer data between PeopleSoft Business Documents and staging tables in the PeopleSoft database. They don’t apply to the “application load” or “application extract” processes, which copy data between the staging tables and the transactional tables.

To define an inbound map

1. In the Application Designer, create the staging tables for the incoming data.

The staging tables are an intermediate step between the PeopleSoft Business Document format and the transactional application tables. So, the structure of the tables is intermediate between the PeopleSoft Business Document format and the structure of the transaction tables.

The record definition for a staging table has three kinds of fields in it:

- Required system fields
- Fields corresponding to the data fields in the PeopleSoft Business Documents
- Extra fields for calculated data needed in the application tables, such as summary values, date stamps, or action codes (optional)

The required system fields are:

ECTRANSID	Transaction ID
ECQUEUEINSTANCE	Instance ID
ECTRANSINOUTSW	I or O, specifying an incoming or outgoing transaction

These fields must be key fields for the tables.

Save the record definitions with the extension **_EC** to identify them as staging tables. Use **SQL Create** to create tables using the record definitions.

2. In the Application Designer, create work records that mimic the structure of the records in the PeopleSoft Business Document.

You need to create a work record for each physical record in the PeopleSoft Business Document file.

One (and only one) field in the work record needs to be designated as holding the Record ID for the document record it mimics. For most records, it's the first field, reflecting the fact that the Record ID in the PeopleSoft Business Document is usually the first data on the line.

The EDI Agent uses this field to match the appropriate work record with each record in the PeopleSoft Business Document. Later in this procedure, you'll specify which Record ID this work record applies to. When the EDI Agent reads a record from the PeopleSoft Business Document, it uses the work record with the same Record ID to read the record's data. If you want, you can add the appropriate Record ID as a default value for the field; the EDI Manager will automatically fill in the correct Record ID at step 9.

The rest of the fields in the work record definition should match the corresponding fields in the PeopleSoft Business Document record. They should be in the same order and should have the same length. Add default values for any fields that have them.

Save the work record with the extension **_WD** to identify it as a work document.

3. From your browser, select PeopleTools, EDI Manager, Use, Inbound Map Definition.

To define a new map definition, select **Add a New Value**, enter a unique value in the dialog box, then click Add.

To update an existing definition, select **Search** or **Find an Existing Value** to select from a list of available codes.

Inbound Map Definition Description page

You use this page to specify to which transaction this map applies.

4. Select the transaction to which this map applies.

All defined inbound transactions appear in the **EC Transaction ID** list.




For information about defining electronic commerce transactions, see the EDI Manager section.

5. Enter a description of the map.

You can enter both a short description (in the **Description** text box) and a long description (in the larger box). These descriptions are for your information only; the system doesn't use them.

6. Copy the definition of a similar map (optional).

If the map you're creating is similar to an existing map, you can copy the details of the existing map definition into this component. Then, you can change just the parts that are different.

Select the existing map in the **Source Map Definition** list box, then click the  button. The system copies the map information from the selected map into the pages.

7. Click the Business Document Layout tab.

You use the Business Document Layout page to describe the format of the PeopleSoft Business Document.

Business Document Layout page

At step 2, you created work record definitions that mimic the structure of the PeopleSoft Business Document. On this page, you specify which work record definition to use for each record in the PeopleSoft Business Document, and what type of data appears in each field.

This page has two areas. The outer area enables you to associate multiple work record definitions with this map. The inner area scrolls through the list of fields in a work record definition.

8. Select one of the work record definitions you created in step 2, then click the lightning bolt button.

Select the record definition from the **Business Doc Record** list box. When you click the lightning bolt button, the EDI Manager copies data from that record definition into the rest of the fields on this page.

9. Specify to which Record ID this work record definition applies.

If the first field in the work record definition isn't the one for Record IDs, use the inner area to scroll to the Record ID field. The EDI Agent uses this field to match the work record definition to the appropriate record(s) in the PeopleSoft Business Document.

- With the field displaying on the page, click the **Row ID** option button in the **Field Value Conversion** group box. A text box appears next to the option.
- In the text box, enter the Record ID of the records whose structure this work record definition mimics. When the EDI Agent finds a record in the PeopleSoft Business Document with this Record ID, it will use this work record definition to parse it.

10. Scroll to the next field in the work record definition.

The inner scroll bar on the page controls the list of record definition fields.

11. Check the location of a field in the record.

If you defined the work record definition at step 2 properly, the correct field information should appear in the text boxes, copied from the work record. You can verify the existing information rather than entering it. If the format of the PeopleSoft Business Document is slightly different than the work record definition, you can modify the formatting on this page to match the document without affecting the work record definition.

The **Seq** text box gives the number of the field in the record. The first field in the record is field 1, the second is field 2, and so on. The Name is the field name from the work record, and the Field Type indicates the data type of the field's data.

The **Start** and **Length** boxes specify the location and size of the field in the PeopleSoft Business Document. The Start position is the number of characters from the beginning of the record to where the first character of this field's data appears. The Length is the number of characters reserved for this field; it's also the number of characters the system will transfer to the staging tables.

For numeric data, the **Dec Positions** is the number of characters to the right of the decimal point. If an incoming PeopleSoft Business Document uses implicit decimal format—that is, if it doesn't include decimal points in this field (in any row)—the EDI Agent will insert them (in all rows) at the position specified in this box. If the data already includes decimal points, the EDI Agent doesn't change them or insert any other decimal points.

For Date data, you also need to enter a Date Fmt and Delimiter. You use these text boxes to specify the format for dates *in this field* in the incoming file. Use standard date formatting conventions, as illustrated in this table.

Date Value	Date Fmt	Delimiter
19961226	YYYYMMDD	N (for None)
1996/12/26	YYYYMMDD	/
DEC-96-26	MMYYDD	-
26-DEC-1996	DDMMYYYY	-
26.12.1996	DDMMYYYY	.

12. Specify what type of data the field contains.

The Special EDI Attribute group box contains a number of option buttons. You select one to specify whether the data in the current field is transaction data or is an EDI control code of some kind. The available options are:

None

The data in the field is normal transaction data.

EC Entity Code	The field gives an Entity Code, which specifies what type of business entity the Trading Partner ID for the current transaction refers to. The Entity Code tells the EDI Agent what table to use to find the list of valid IDs.
EC Queue Control Number	The data in this field is a unique identifying number that you want to store in the EDI Agent queue so that you can refer to it when reviewing EDI audit history. For example, in a purchase order, you might identify the PO Number as the EC Queue Control Number.
Pri Evt Cd	The field identifies the Primary Event Code, sometimes called the Purpose Code. The EDI Agent uses the fields identified as Event Codes to determine the appropriate Action Code (see step 20). If you identify a Primary Event Code, you must also identify a Secondary Event Code in the same record.
Sec Evt Cd	The field identifies the Secondary Event Code, sometimes called the Transaction Code. The EDI Agent uses the fields identified as Event Codes to determine the appropriate Action Code. If you identify a Secondary Event Code, you must also identify a Primary Event Code in the same record.



For more information about Entity Codes, Event Codes, and Action Codes, see the EDI Manager section.

13. Specify how the EDI Agent needs to convert the data as it copies data from the field into the staging tables.

The EDI Agent can perform certain kinds of data conversion as it copies data from the PeopleSoft Business Document into the staging tables. The **Field Value Conversion** options are:

None	The EDI Agent copies the data exactly as it appears in the PeopleSoft Business Document field.
TP Convert	The EDI Agent converts data from this field according to the rules defined in a conversion type definition. When you select this option, a drop-down list box appears so that you can select which conversion type definition to use.
Row ID	Use this option for the first field in the work record definition only. See step 9 for details.

Convert

The EDI Agent converts the data as specified in the From and To boxes. When the field in the PeopleSoft Business Document contains the value in the From box, the EDI Agent converts it into the value in the To box. This option includes a scroll bar, and you need to enter a row for each possible value in the field.



You can perform data conversion at several points during processing. Use the **Convert** option for conversions that apply to this map only and are valid for all trading partners.

14. Repeat steps 10 through 13 for each field in the work record definition.

15. Repeat steps 8 through 14 for each record in the PeopleSoft Business Document.

To associate another work record definition with this map, put the cursor in the Business Doc Record list box and press the F7 key.

16. Click the **Target Records** tab.

You use the Target Records page to specify into which staging tables to copy the transaction data.

Target Records page

The page has three areas. The outer area enables you to move between the work record definitions you associated with this mapping on the previous page. For each work record definition, you can specify one or more staging tables to copy data into; that's what the

second area is for. The third (innermost) area moves through the list of fields in the staging table's record definition.

17. Select a work record definition.

Use the outer scroll bar to move between the work record definitions you added to this mapping on the previous page. The record definition name appears in the **File Layout Model** field.

18. Select the staging table to copy data into, then click the lightning bolt button.

In the **Layout Record** list, select the record definition for the table you want to copy transaction data to. Click the lightning bolt button to copy field information from the record definition into the page.

19. Select a field in the staging table record definition.

Use the inner scroll bar to move between fields.

The **Sequence** text box gives the number of the field in the record. The **Field Name** is the field name from the record definition, and the **Field Type** indicates the data type of the field's data. All of this information comes from the record definition you selected.

20. Specify what data to copy into this field.

In the **Set Field Equal To** area, select the value you want the EDI Agent to copy into the selected field.

- To copy the value from a field in the PeopleSoft Business Document, select the **File Field Value** option button. Enter the name of the field whose value you want to copy in. Use the field name from the work record definition that appears as the File Layout Model at the top of the page.
- To enter a default value into the table, select the **Default Value** option button and enter the desired value in the text box that appears next to it.



You can specify *both* a file field to copy data from and a default value. If you do, the EDI Agent enters the default value into the staging table only when the specified file field doesn't have a value.

- To enter a value that the EDI Agent calculates as it copies data into the staging table, select the **EC Agent Calc'd** option button, then pick a calculation option from the drop-down list that appears next to it. The table below describes the available calculation options.

<i>Calculation Option</i>	<i>Value Written in the Field</i>
Action Code Conversion	<p>An Action Code.</p> <p>The EDI Agent gets the values from the fields in this record labeled Primary Event Code and Secondary Event Code, looks up the combination in the partner profile for the trading partner, and inserts the appropriate Action Code.</p>

<i>Calculation Option</i>	<i>Value Written in the Field</i>
Apportion Parent Value	<p>A percentage of the value from the parent line.</p> <p>The EDI Agent takes the numeric value from the preceding parent line, divides it by the number of child lines, and puts the resulting value in this field on each child line.</p> <p>You have to fill out the Related Record Info to identify the parent line.</p>
Average Summary Value	<p>The average value from a particular field.</p> <p>The EDI Agent adds up the values in this field from all occurrences of this record reporting to the same parent line, divides by the number of records, and puts the result in this field.</p> <p>You have to fill out the Related Record Info to identify the field whose average you want.</p>
Business Document Level External TPID	<p>The Trading Partner ID of the trading partner that generated the EDI transaction.</p> <p>The EDI Agent gets the Trading Partner ID from the first record in the PeopleSoft Business Document.</p>
Business Document Level Internal TPID	<p>The Trading Partner ID of the internal group to which the EDI transaction is addressed.</p> <p>The EDI Agent gets the Trading Partner ID from the first record in the PeopleSoft Business Document.</p>
Current Date	The date on which the EDI Agent processes the PeopleSoft Business Document.
Current Date and Time	The date and time when the EDI Agent processes the PeopleSoft Business Document.
EC Queue Instance	<p>The system-generated Queue Instance ID given to this PeopleSoft Business Document.</p> <p>Note: To create a valid mapping, you must map this value into the ECQUEUEINSTANCE field.</p>
EC Transaction ID	<p>The Transaction ID from the first record in the PeopleSoft Business Document.</p> <p>Note: To create a valid mapping, you must map this value into the EDTRANSID field.</p>
File Name/File ID	The name of the PeopleSoft Business Document file.
Incremented Key Sequence Number	<p>A system-generated sequence number.</p> <p>The EDI Agent increments the number for each child row of a particular parent. It starts at 1 again when it reaches a new occurrence of the parent line.</p> <p>You have to fill out the Related Record Info to identify the</p>

<i>Calculation Option</i>	<i>Value Written in the Field</i>
	parent line.
Inherit Parent Value	A value copied directly from the parent line. You have to fill out the Related Record Info to identify the parent line.
Maximum Summary Value	The maximum value of a field. The EDI Agent checks all the occurrences of a specified record and field, and copies the maximum value into this field. You have to fill out the Related Record Info to identify the field whose maximum you want.
Minimum Summary Value	The minimum value of a field. The EDI Agent checks all the occurrences of a specified record and field, and copies the minimum value into this field. You have to fill out the Related Record Info to identify the field whose minimum you want.
Operator ID	The Operator ID of the user who started the EDI Agent.
Process Instance	The system-generated Instance ID from the Process Scheduler.
Run Control ID	The Run Control ID of the run control used to start the EDI Agent.
Total/Accumulate Summary Value	The total of the values in a field. The EDI Agent adds together the values from a specified field in all occurrences of a record reporting to the same parent line and puts the result in this field. You have to fill out the Related Record Info to identify the field whose total you want.
Trading Partner Conversion	The customer, vendor, or business unit associated with a Trading Partner ID.

21. Enter the related record information (for some calculated options only).

When you select some of the calculation options in step 20, three drop-down list boxes appear in the **Related Record Info** area. You use these list boxes to identify the Row, Record, and Field to use in the calculation.

For example, if you selected the **Total/Accumulate Summary Value** option, the EDI Agent will add up the values from the specified row, record, and field.



If you selected the **Increment Key Sequence Number** option, only the **Row** list box appears. You don't need to select a record or field.

22. Repeat steps 19 through 21 for each field in the staging table.

To create a valid mapping, you must enter the appropriate values in the three required system fields:

ECTRANSID	Use the EC Transaction ID option to have the EDI Agent put the Transaction ID in this field.
ECQUEUEINSTANCE	Use the EC Queue Instance ID option to have the EDI Agent put the Instance ID in this field.
ECTRANSINOUTSW	Enter a Default Value of I in this field, to specify an inbound map definition.

23. Repeat steps 18 to 22 for each staging table you want to copy data into.

24. Repeat steps 17 to 23 for each work record definition.

25. Save the map.

To define an outbound map

1. From your browser, select PeopleTools, EDI Manager, Use, Outbound Map Definition.

To define a new map definition, select **Add a New Value**, enter a unique value in the dialog box, then click Add.

To update an existing definition, select **Search** or **Find an Existing Value** to select from a list of available codes.

This page appears.

Outbound Map Definition Description page

2. Select the transaction to which this map applies.

All defined outbound transactions appear in the **EC Transaction ID** list.



For information about defining electronic commerce transactions, see the EDI Manager section.

3. Enter a description of the map.

You can enter both a short description (in the **Description** text box) and a long description (in the larger box). These descriptions are for your information only; the system doesn't use them.

4. Copy the definition of a similar map (optional).

If the map you're creating is similar to an existing map, you can copy the details of the existing map definition into this component. Then, you can change just the parts that are different.

Select the existing map in the **Source Map Definition** list box, then click the  button. The system copies the map information from the selected map into the pages.

5. Click the **Source Records** tab.

You use the Source Records page to specify which staging tables to retrieve data from to create PeopleSoft Business Documents.

Source Records page

6. Select the record definition for extracting data from the staging table.

In the **Source Record** list box, select the record definition that retrieves the data you want from the staging tables. It could be the record definition used to define the table or a view that retrieves just the data you want.

You want to add lines to the PeopleSoft Business Document in a logical unit of work order—header information before detail line information, parent lines before child lines, and so on.



You don't need to explicitly create a control record for the PeopleSoft Business Document (the 999 or 998 record). The EDI Agent creates one automatically from information in the queue that stores pending outbound transactions.

7. Select a parent record (child lines only).

If the record definition you selected in step 6 retrieves data for a record that's related to a preceding parent record—such as the detail line under a header—pick the record definition for the parent record in the Parent Record list.

The parent record is typically the record definition you used for the previous row in the outbound mapping.

8. Enter the SQL WHERE clause to specify which rows to retrieve.

When you're creating a header line, the **WHERE clause** needs to select the header information for a single transaction. You identify a single transaction using the Transaction ID and unique Queue Instance ID. The WHERE clause should look like this:

```
WHERE ECTRANSID = $ECTRANSID$ AND ECQUEUEINSTANCE = $ECQUEUEINSTANCE$
```

The two values inside dollar signs are system variables that the EDI Agent replaces with the Transaction ID and Queue Instance ID respectively of the first transaction in the pending outbound queue. This WHERE clause has the effect of only retrieving the data for that one transaction.

If you're extracting the data for child lines, the WHERE clause must include a link to the parent record, to tell the EDI Agent how to join the tables. The WHERE clause needs to include a statement of this form:

```
WHERE fieldname = $record.fieldname$
```

The first *fieldname* is a field from the source record definition; *record.fieldname* is a field in the parent record definition. You must include the dollar signs around *record.fieldname*.

9. Repeat steps 6 to 8 to specify the record definition for each record to go into the PeopleSoft Business Document.
10. Click the Target Record Doc tab.

You use the Target Record Doc Layout page to specify how the EDI Agent formats the PeopleSoft Business Document.

The screenshot shows the 'Target Business Doc Layout' page. The 'EC Map ID' is 'TNT', 'Trans ID' is 'GENERALOUT', and 'Descr' is empty. The 'File Row ID' is '000'. The 'Model File Layout' is empty. The 'Target Record Field Info' section has 'Sequence' at 0, 'Start' at 0, 'Length' at 0, 'Field Type' as 'CHAR', and 'Convert to Upper Case' is unchecked. The 'Conversion Processing' section has 'Source Field' selected. The bottom navigation bar includes 'Save', 'Previous tab', 'Next tab', 'Add', 'Update/Display', and 'Correct History' buttons.

Target Business Document Layout page

For each of the source record definitions you added on the previous page, you identify an associated work record definition that specifies how the EDI Agent writes the staging table data into the PeopleSoft Business Document.

This page has two scroll bars. The outer scroll bar enables you to scroll between the source record definitions you added on the previous page; they are identified by the **File Row ID** the system assigned when you added them. The inner scroll scrolls among the fields in the work record definition you associate with the source record definition.

11. Select the work record definition that specifies the format of the output record, then click the lightning bolt button.

The **File Row ID** field identifies a source record definition from the previous page. In the **Model File Layout** list box, choose the work record definition that the EDI Agent will use to write data from the source record into the PeopleSoft Business Document.

When you click the lightning bolt button, the system copies field information from the work record definition into the page. The Target Record Field Info group box displays information about where in the record it will write each field's data.

Depending on the data type of the record field, the Target Record Field Info box offers additional formatting options, so that you can format the data in the outgoing PeopleSoft Business Document.

- For any field, you can enter in the Strip Chars box one or more characters you don't want the Outbound EDI Agent to include when it copies data into the outbound transaction file. For example, you may want to "strip" the hyphens from phone numbers and Social Security Numbers, or the decimal from a currency amount. When the Outbound EDI Agent writes data from a field into an outbound transaction file, it will first remove any and all instances of the characters in the **Strip Chars** text box. You can include any number of characters in the **Strip Chars** box.



The **Strip Chars** text box contains a *list* of the individual characters to remove, not a *string* to remove. For example, if the field data is 455-67-8898 and the **Strip Chars** text box has a hyphen in it, the EDI Agent writes 455678898 to the file. If the **Strip Chars** text box has the two characters -5 in it, the EDI Agent writes the same field data to the file as 4678898.

- For *character fields*, you can click the **Convert to Upper Case** check box to tell the Outbound EDI Agent to write the contents of the field in all upper case letters.
- For *character fields or numeric fields*, you can use the **Pad Character** text box to specify a character for the Outbound EDI Agent to use to pad data so it is the full length of the field. For character fields, the Outbound EDI Agent adds the specified character to the left of the existing field information. For numeric fields, the most typical use is to pad a number with initial zeros.
- For *date fields*, you can specify in the **Date Fmt** and **Delimiter** text boxes how you want to format dates in this field in the Business Document. Use standard date formatting conventions, as illustrated in this table.

<i>Date Value</i>	<i>Date Fmt</i>	<i>Delimiter</i>
19961226	YYYYMMDD	N (for None)
1996/12/26	YYYYMMDD	/
DEC-96-26	MMYYDD	-
26-DEC-1996	DDMMYYYY	-
26.12.1996	DDMMYYYY	.

12. Specify what data you want the EDI Agent to write in the field.

In the **Conversion Processing** area, select the value you want the EDI Agent to write into the selected field.

- To copy the value from a field in the source record, select the **Source Field** option button. Enter the name of the field whose value you want to copy in.



You need to specify a source field for all fields in the work record definition, except those for which you provide a default value. You must pick a source field even if you select one of the other options as well.

- To enter a default value into the document, select the **Default Value** option button and enter the desired value in the text box that appears next to it.
- You can specify both a file field to copy data from and a default value. If you do, the EDI Agent enters the default value into the document only when the specified field doesn't have a value.
- If you select **TP Conversion**, the EDI Agent converts data from the specified source field according to the rules defined in a conversion type definition. When you select this option, a drop-down list box appears so that you can select which conversion type definition to use.
- If you select **Convert**, the EDI Agent converts the data in the source field as specified in the **From** and **To** boxes that appear below it. When the field in the PeopleSoft Business Document contains the value in the **From** box, the EDI Agent converts it into the value in the **To** box. This option includes a scroll bar, and you need to enter a row for each possible value in the field.
- To enter a value that the EDI Agent calculates as it copies data into the staging table, select the **Agent Value** option button, then pick a calculation option from the drop-down list that appears next to it. The table below describes the available calculation options.

<i>Calculation Option</i>	<i>Value Written in the Field</i>
Action Code Conversion	An Action Code. The EDI Agent gets the values from the fields in this record labeled Primary Event Code and Secondary Event Code, looks up the combination in the partner profile for the

	trading partner, and inserts the appropriate Action Code.
Current Date	The date on which the EDI Agent creates the PeopleSoft Business Document.
Current Datetime	The date and time when the EDI Agent creates the PeopleSoft Business Document.
EC Entity Code Flag	Flags this field as an Entity Code field to use in determining trading partner conversion logic. The values specify whether the entity is a customer, vendor, or business unit.
EC Trans ID	The Transaction ID.
Incremented Key Sequence Nbr	A system-generated sequence number. The EDI Agent increments the number for each child row of a particular parent. It starts at 1 again when it reaches a new occurrence of the parent line.
Secondary Event Code	The Secondary Event Code from the Action Code conversion.
Trading Partner Conversion	The Trading Partner ID associated with the customer, vendor, or business unit.

13. Move to the next field in the record definition and repeat step 12.

14. Scroll to the next source record definition and repeat steps 11 to 13.

15. Save the map.

16. Run the Outbound EDI Agent Preparer to create the Outbound EDI Agent SQC.

After you create or modify outbound maps, you need to run a special compilation process that prepares the maps for use by the EDI Agent. For information about running this process, see *Monitoring EDI Processing*.



Run the Outbound EDI Agent Preparer only after you've made all your changes to outbound maps. You only need to run it once to prepare all outbound maps.

Creating Map Profiles

When you add a trading partner, you assign to it a *map profile*, which lists the electronic commerce maps that the EDI Agent can use to process transactions from the partner. The map profile serves two purposes:

- It restricts the trading partners access to transactions they aren't authorized to exchange with your company.
- By assigning different map profiles to different trading partners, you can get the EDI Agent to

process the “same” transaction differently for different partners.

All you do to define a map profile is build a list of the maps you want to make available to partners with that profile. Then, when you add a new trading partner, you select which map profile to use for that partner.

To create a map profile

1. From your browser, select PeopleTools, EDI Manager, Use, Data Mapping Profile Defn.

To define a new map definition, select **Add a New Value**, enter a unique value in the dialog box, then click Add.

To update an existing definition, select **Search** or **Find an Existing Value** to select from a list of available codes.

Ec Map Profile Def

EC Map Profile ID: GENERALMP

Descr: General Map Profile

Map Assignment Find | View All First 1 of 1 Last

EC Map ID	EC Transaction ID
	Add Delete

Save Return to Search Add Update/Display

EC Map Profile Definition page

You use this page to specify which maps are available for partners with this Map Profile ID.

2. Enter a description of the map profile in the **Descr** text box.
3. Add the maps you want partners with this map profile to use.

Select the **Map ID** from the **EC Map ID** list. Repeat this step until you’ve added maps for all the transactions partners with this profile are able to perform.

Don’t forget to add both inbound and outbound maps.

4. Save the page.

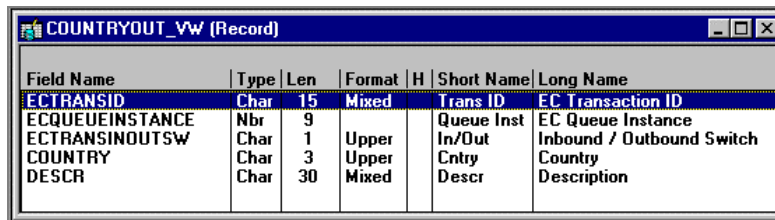
Using the EDI Manager for General Data Extraction

The primary purpose of the EDI Manager is to create PeopleSoft Business Documents, which contain business transaction data and are subsequently translated into X.12 or EDIFACT format and transmitted to a trading partner. However, you can also use it as a general tool for extracting data from database tables into a text file. This section provides an overview of this process.

To create a text file from PeopleSoft database data

1. Using the Application Designer, create a view that extracts the data you want and includes three EDI control fields.

For example, suppose you want to print a list of the countries in COUNTRY_TBL. You need to create a view with the three EDI control fields described below, plus the fields you want from COUNTRY_TBL. The view would look something like this:



Field Name	Type	Len	Format	H	Short Name	Long Name
ECTRANSID	Char	15	Mixed		Trans ID	EC Transaction ID
ECQUEUEINSTANCE	Nbr	9			Queue Inst	EC Queue Instance
ECTRANSINOUTSW	Char	1	Upper		In/Out	Inbound / Outbound Switch
COUNTRY	Char	3	Upper		Cntry	Country
DESCR	Char	30	Mixed		Descr	Description

The EDI control fields must be the first three fields in the view. The fields are:

EDITRANSID

The Transaction ID for the transaction you want to perform. You'll probably want to define a special Transaction ID for this data extraction transaction. For details, see Defining EDI Transactions.

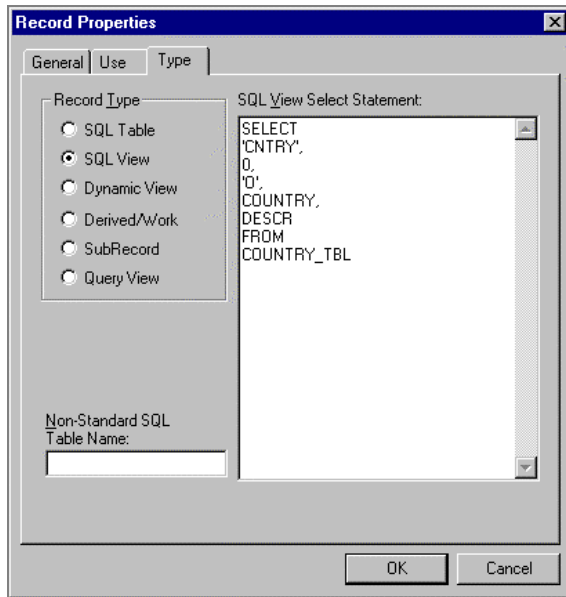
ECQUEUEINSTANCE

An ID field you can use to track individual transactions while they're in the EDI Agent queue awaiting processing. If you don't need to track individual transactions, you can use the same value in this field for all transactions.

ECTRANSINOUTSW

A field specifying whether a transaction is incoming (I) or outgoing (O). For outgoing transactions, this value is always O, so the view definition can specify 'O' as the value for this field.

The SELECT statement for the view looks something like this:



In this example, the Transaction ID is CNTRY, the queue instance ID is always 0, and the inbound/outbound switch is always 'O' (Outbound). Since the queue instance will be the same for all records, the EDI Agent will process them as a single transaction.



For this example to work, you must use the EDI Manager to define the Transaction ID CNTRY.

2. Create a record definition that specifies the layout of the PeopleSoft Business Document file.

This record definition typically consists of the ECFILEROWID field and all the fields from the table.

COUNTRYOUT_WD (Record)						
Field Name	Type	Len	Format	H	Short Name	Long Name
ECFILEROWID	Char	3	Upper		Row ID	File Row ID
COUNTRY	Char	3	Upper		Cntry	Country
DESCR	Char	30	Mixed		Descr	Description

3. Define the outbound map.

In the example, you would specify CNTRY as the **Transaction ID** on the first page in the Outbound Map Definition component. On the second page, the **Source Record** is the view you created in step 1 and the **Where Clause** is:

```
WHERE ECTRANSID = 'CNTRY'
```

On the third page, the **Model File Layout** is the record definition you created in step 2. Once you've selected the record definition from the list box, click the lightning bolt button to copy the field information into the page. You can use the default values for all the fields except ECFILEROWID. For ECFILEROWID, select **Default** in the **Conversion Processing** group box and enter whatever value you want to appear in the first column of the text file.

4. Run the Outbound Driver Preparation process.



For information about running this process, see Preparing Outbound Maps.

5. Add an entry to the ECQUEUE table.

To extract your data, you need to add a record to the ECQUEUE table. The EDI Agent uses this table to determine what transactions are ready to process. The fields in the table are described below.

ECTRANSID	The Transaction ID, which tells the EDI Agent which map definition to use. In the example, the Transaction ID is CNTRY.
ECQUEUEINSTANCE	The Instance ID. The example uses a single Instance ID—0 (zero)—for all the data.
ECTRANSINOUTSW	The inbound/outbound switch. Extracting data is an outbound transaction, so the value is O (the letter O).
ECBUSDOCID	The Business Document ID. Since you are not creating an EDI transaction, you can specify 0 in this field.
ECQUEUESTATUS	The status of the current record. The EDI Agent processes records with the status L.

The remaining fields in the table—BUSINESS_UNIT, ECENTITYCD_BU, ECCUSTVNDRVAL, and ECENTITYCD_EXT—all relate to trading partner information. Since you are not creating an EDI transaction, you can use generic partner information. The EDI Manager includes a set of default values especially for this purpose. Use the value GENR in all these fields.

6. Schedule the Outbound EDI Agent to run.

The Outbound EDI Agent extracts your data, and processes any other transactions on the ECQUEUE table with the status L.

When the Outbound EDI Agent extracts your data, it changes the status field ECQUEUESTATUS to P (Processed). To repeat the extraction, change the status back to L.

CHAPTER 9

Monitoring EDI Processing

You will use the *Process* and *Inquire* pages in the EDI Manager on an ongoing basis to keep your electronic commerce transactions flowing smoothly.

This chapter explains how to:

- Prepare outbound maps for use by the EDI Agent
- Schedule EDI Agents to run
- Review and correct EDI processing errors
- Review processing history



You must run the outbound map preparation process before you can process any outgoing EDI transactions.

Managing EDI Agents

EDI Agents copy EDI transaction data between PeopleSoft Business Documents and the PeopleSoft database, using the maps you defined in the previous chapter. There are two types of EDI Agents:

- Inbound EDI Agents, which process incoming transactions by copying data from PeopleSoft Business Documents into the database
- Outbound EDI Agents, which create PeopleSoft Business Documents from transaction data in the EDI staging tables

Creating EDI maps doesn't start an EDI Agent running. You have to submit a Process Scheduler request that tells the system when, where, and how often to run each EDI Agent.

This section describes how to start one or more EDI Agents to process your incoming and outgoing EDI transactions. It also explains the procedure you need to complete to prepare outbound maps for use by the EDI Agent.

Run Controls

All the procedures described in this section involve scheduling tasks to run using the Process Scheduler. To schedule a process, you need to tell the system when and where you want the process to run. For example, you might tell it to run the inbound EDI Agent at five o'clock, or every day at five o'clock, or on your workstation right away. Depending on the process, you may need to specify other parameters as well, such which documents or transactions to process.

A *run control* is a database record that provides values for these settings. Instead of entering the same values each time you schedule a process, you create (and save) a run control with those settings. The next time you schedule the process, you select the run control and the system fills in the settings.

So, when you select any of the **Process** menu options, the EDI Manager asks for a Run Control ID. To use an existing run control record, select Update/Display mode; to create a new one, select Add mode.

Preparing Outbound Maps

When you create an outbound map definition, you define what data the EDI Agent needs to extract from the staging tables. In order to select that data from the tables, the EDI Agent needs to execute SQL statements whose details depend on the map definition.

Whenever you create or modify an outbound map definition, you need to run a preparation process that generates the appropriate SQL statements and compiles them into a format that the EDI Agent can use. This process creates an SQC file.



Note. You must run the Outbound Driver Preparation before you can process any outbound EDI transactions. PeopleSoft doesn't deliver a compiled version of the drivers; you have to compile it for your system.

To prepare outbound maps for the EDI Agent

1. From your browser, select PeopleTools, EDI Manager, Process, Outbound Driver Preparation.

To define a new run control, select **Add a New Value**, enter a unique value in the dialog box, then click Add.

To update an existing run control definition, select **Search** or **Find an Existing Value** to select from a list of available codes.

Run Parameters

Run Control ID: HRRpts [Report Manager](#) [Process Monitor](#) [Run](#)

ECOUTMAP.SQC Directory:

Map Selection Criteria

Trans ID: [Select](#) [Unselect](#) [Refresh](#)

[View All](#) First 1 of 1 Last

Select	Transaction ID	Map ID	Description
<input type="checkbox"/>			

[Save](#) [Return to Search](#) [Add](#) [Update/Display](#)

Outbound Driver Preparation page

- Specify where to put the SQC that contains the outbound map definitions.

In the **ECOUTMAP.SQC Directory** text box, enter the directory where you want the Process Scheduler to put the SQC. The directory needs to be in the SQR search path of the workstation or server that will run the outbound EDI Agent.

- Click the Run button.

The Process Scheduler Request page displays.

Process Scheduler Request

User ID: PTDMO Run Control ID: CAVEMAN

Server Name: Run Date: 07/14/2000

Recurrence: Run Time: 12:33:33PM

Time Zone: [Reset to Current Date/Time](#)

Process List

Select	Description	Process Name	Process Type	*Type	*Format
<input checked="" type="checkbox"/>	Outbound EC Agent Prep	ECPREP	SQR Report	Web	PDF

[OK](#) [Cancel](#)

Process Scheduler Request page

- Specify the server where you want the SQR compiler to run.
- Specify when you want the process to run.

You'll probably want to run the preparation process just once (then run it again after changing or adding maps). Enter the date and time when you want to run it in the **Date** and **Time** boxes. They default to the current date and time.

6. Click the **OK** button to run (or schedule to run) the outbound map preparation.

Starting EDI Agents

If your system processes EDI transactions, you'll usually want to have two EDI Agents running on a regular schedule: one inbound agent and one outbound agent. You can also start other EDI Agent instances for special purposes, such as processing the corrected version of a PeopleSoft Business Document that failed the first time.

If errors occur at any point during EDI processing, the EDI Agent sends a worklist item to the role user or users assigned to the EDI COORDINATOR role. For more information about reviewing the errors, see Viewing the EDI Audit Trail, later in this chapter.

To schedule the Inbound EDI Agent to run

1. From your browser, select PeopleTools, EDI Manager, Process, Inbound EC Agent.

To define a new run control, select **Add a New Value**, enter a unique value in the dialog box, then click Add.

To update an existing run control definition, select **Search** or **Find an Existing Value** to select from a list of available codes.

The screenshot displays the 'Run Control Parameters' window for the 'CAVEMAN' Run Control ID. At the top, there are links for 'Report Manager' and 'Process Monitor', and a 'Run' button. The main area is divided into several sections: 'Run Option' with radio buttons for 'File List Driven' (selected), 'Single File', and 'Single Instance'; 'Inbound Agent Parameters' with text boxes for 'File List Path' and 'File List Name'; 'Force Profile' with radio buttons for 'Do Not Force (998 or 999 in file)', 'Force with Map Information (998)', and 'Force with Partner Information (999)'; 'Inbound Agent Forced Parameters' which is currently empty; and 'File Options' with checkboxes for 'Suppress Rowid' and 'Comma Separated Format'. At the bottom, there are buttons for 'Save', 'Return to Search', 'Add', and 'Update/Display'.

Inbound EC Agent page

You use this page to specify how you want the Inbound EDI Agent to process its files.

2. Select the Run Option that the EDI Agent is to use to determine which documents or transactions it is to process.

The EDI Agent can process all pending PeopleSoft Business Documents, one specific PeopleSoft Business Document, or a single transaction. In the **Run Option** area, select the appropriate option button:

File List Driven	The EDI Agent processes all the PeopleSoft Business Documents whose filenames appear in a list file that the EDI translator software creates. The filenames must include their complete paths.
Single File	The EDI Agent processes all the transactions in a single specified PeopleSoft Business Document.
Single Instance	The EDI Agent processes a single transaction.



The **Single Instance** option is only available for transactions that the EDI Agent has already attempted to process. The transaction needs to have an EC Queue Instance ID.

3. Identify the file or transaction you want the EDI Agent to process.

When you select a **Run Option**, the **Inbound Agent Parameters** area updates to show the fields that are relevant to the selected option.

If you selected **File List Driven**, enter the directory that contains the PeopleSoft Business Document files and the list file in the **File List Path** text box, and the filename of the list file in the File List Name box.

If you selected **Single File**, enter the directory that contains the file in the Single File Path text box, and the PeopleSoft Business Document filename in the **Single File Name** box.



If the EDI Agent will run on a server running MVS, it will ignore the **File List Path** or **Single File Path** setting. Instead, it will look in the JCL for a DDNAME matching the specified file name. The **File List Path** option is not very useful for MVS platforms, because there has to be a DDNAME for each file in the list.

If you selected **Single Instance**, select the **Business Document ID**, **Transaction ID**, and **Queue Instance ID** of the transaction you want to process, in that order.

Run Control Parameters		
Run Control ID: CAVEMAN		Report Manager Process Monitor Run
Run Option <input type="radio"/> File List Driven <input type="radio"/> Single File <input checked="" type="radio"/> Single Instance	Inbound Agent Parameters EC Business Document ID: <input type="text"/> <input type="button" value="Q"/> EC Transaction ID: <input type="text"/> <input type="button" value="Q"/> EC Queue Instance: <input type="text"/> <input type="button" value="Q"/>	
Force Profile <input type="radio"/> Do Not Force (998 or 999 in file) <input type="radio"/> Force with Map Information (998) <input type="radio"/> Force with Partner Information (999)	Inbound Agent Forced Parameters <div style="border: 1px solid black; height: 100px; width: 100%;"></div>	File Options <input type="checkbox"/> Suppress Rowid <input type="checkbox"/> Comma Separated Format
Save Return to Search		Add Update/Display

Single Instance Run Option page

4. Specify how the Inbound Agent determines which map definition to use.

By default, the Inbound Agent looks for records with the Row ID 999 to determine what type of transaction it's processing, and therefore which map definition to use. However, you can also specify a transaction map or trading partner for *all* transactions this Inbound EDI Agent processes. Using this feature, the Inbound EDI Agent can process files that don't include control records.

If you select the **Do Not Force** option, the Inbound EDI Agent will look for a control record at the beginning of each transaction in the incoming data file. You don't need to complete any other parameters.

If you select **Force with Map Information**, the Inbound Agent Forced Parameters box displays list boxes so you can select the transaction map to use for all transactions in the incoming file.

Force Profile	Inbound Agent Forced Parameters
<input type="radio"/> Do Not Force (998 or 999 in file) <input checked="" type="radio"/> Force with Map Information (998) <input type="radio"/> Force with Partner Information (999)	EC Map ID: <input type="text"/> <input type="button" value="Q"/> EC Profile ID: <input type="text"/> <input type="button" value="Q"/> EC Convert Profile ID: <input type="text"/> <input type="button" value="Q"/>

If you select **Force with Partner Information**, the box displays list boxes so you can select the partner information to use for all transactions in the incoming file.

Force Profile	Inbound Agent Forced Parameters
<input type="radio"/> Do Not Force (998 or 999 in file)	External Entity Code: <input type="text"/>
<input type="radio"/> Force with Map Information (998)	Forced Trans Id: <input type="text"/>
<input checked="" type="radio"/> Force with Partner Information (999)	External Trading Partner Id: <input type="text"/>
	Internal Alias Entity Code: <input type="text"/>
	EC Alias Trading Partner ID: <input type="text"/>

5. Select the File Option the EDI agent is to use to display file data.

File Options
<input type="checkbox"/> Suppress Rowid
<input type="checkbox"/> Comma Separated Format

Suppress Rowid:

Use this option to map homogenous files to the same table. A rowid of '000' must be specified in the map definition, however, the flat file need not have a record identifier.

Comma Separated Format:

Use this option to import files of variable length fields into PeopleSoft tables. This option is only available when the profile is forced with either 999 or 998 information.



The Separator should appear between each field in the file. The Delimiter surrounds each field.

6. Click the **Run** button.

The **Process Scheduler Request** page displays.

Process Scheduler Request					
User ID:	PTDMO		Run Control ID:	NEW_ID	
Server Name:	<input type="text"/>	Run Date:	<input type="text" value="07/14/2000"/>		
Recurrence:	<input type="text"/>	Run Time:	<input type="text" value="12:57:36PM"/>		
Time Zone:	<input type="text"/>	<input type="button" value="Reset to Current Date/Time"/>			
Process List					
Select	Description	Process Name	Process Type	*Type	*Format
<input checked="" type="checkbox"/>	Inbound EC Agent	ECIN0001	SQR Report	Web	PDF
<input type="button" value="OK"/>		<input type="button" value="Cancel"/>			

Process Scheduler Request page

7. Specify on what server you want the agent to run.



In a production workflow, you'll usually want to run the EDI Agent on a dedicated server.

8. Specify when—and how often—you want the agent to run.

If you're running the EDI Agent once, enter the date and time when you want to run it in the **Date** and **Time** boxes. They default to the current date and time.

If you're running the agents on a server, the Process Scheduler can run them once or periodically on a specified schedule. You use the **Run Recurrence** box to specify when and how often to run the database agents.

By default, the system runs the agents **Once**, but you'll usually want to run the more frequently. To run them more than once, select a different *recurrence name* from the list box. A recurrence name provides a running schedule; it specifies when to run the agent for the first time and how often to run it after that.

To see the schedule associated with a recurrence name, select it in the list box and click the **View** button. To add a new recurrence name, type a name into the text box and click the **New** button. See Process Scheduler for details.

If you selected a **Run Recurrence** other than **Once**, the recurrence definition sets the run date and time. It overrides any **Run Date/Time** you set.



The **Run Output** option is not relevant for the EDI Agent.

9. Click the **OK** button to run (or schedule to run) the EDI Agent.

To schedule the Outbound EDI Agent to run

1. From your browser, select PeopleTools, EDI Manager, Process, Outbound EC Agent.

To define a new run control, select **Add a New Value**, enter a unique value in the dialog box, then click Add.

To update an existing run control definition, select **Search** or **Find an Existing Value** to select from a list of available codes.

Outbound EC Agent page

You use this page to specify how you want the Outbound EDI Agent to write data to the PeopleSoft Business Document files.

2. Specify which transactions you want the EDI Agent to process.

To have the EDI Agent process all pending transactions, leave all three check boxes unselected. A pending transaction is one that an application extract process has added to the staging tables, and that has a status of *L* (Loaded).

To process all transactions of a particular type, select the **EC Trans ID** check box, then pick the Transaction ID from the drop-down list that appears next to it.

To process all transactions from a particular business unit, select the **Business Unit** check box and enter the Business Unit ID in the text box that appears next to it.

To process all transactions for a particular trading partner, select the **Vendor/Customer** check box, then enter the Vendor ID, Customer ID, or other internal ID for the desired partner.

You can select multiple check boxes. For example, you could process all outgoing invoices for a particular vendor by selecting both the **EC Trans ID** and **Vendor/Customer** check boxes.

3. Specify how you want to format the outgoing transaction file.

By default, the Outbound EDI Agent adds a record with the Row ID 999 at the beginning of each transaction. This record identifies the transaction type in a way that is recognized by the PeopleSoft Inbound EDI Agent. If you select the **Suppress EC 999 Record** check box, the Outbound EDI Agent will not include this record in the outgoing file.

If you don't want to include the Row ID for each record in the outgoing file, select the **Suppress Rowid** check box.

By default, the Outbound EDI Agent creates outbound transactions files in which the data for each field appears in the fixed column position specified in the outbound map definition. If you select the **Comma Separated Format** check box, the Outbound EDI Agent instead creates transaction files in which the fields are separated by a comma or other character rather than always appearing in a specified column position. The comma separated option allows you to select the file options to read in flat file data that is comma-delimited.



The Comma Separated option is only available if you used forced agent mapping. Selecting the Suppress Rowid option allows you to load data without requiring a rowid identifier. However, it is required that your map definition contains a rowid even if you are suppressing it in the run control setup. The rowid must be 000.

The **CV Separator** is the character the EDI Agent will use to separate the fields in a record. The **CSV Delimiter** is the character the EDI Agent uses to mark the beginning and end of a data value. In the text below, the separator is a comma, and the delimiter is a single quote.

'WOW','23.23','Big Deal'



When you use comma-separated format, the Outbound EDI Agent ignores the column positions specified in the outbound map definition. It adds data to the outbound file in the exact order in which the fields appear in the map definition. You must make sure the map definition specifies the fields in the order you want them to appear; that is, you must make sure the order of the rows in the map definition matches the order you want the fields to appear in the file.

Single Document File

This option allows the results of the EDI to be written to a single Output File. The business document id is incremented as though multiple documents were created. The file may contain multiple 999 records.

Keep Queue Status

The ECQUEUE. ECQUEUESTATUS field will not be updated at the completion of the execution. This allows the record in ECQUEUE to be processed multiple times.

Message Header/Footer

Message Header and Footer Information may be specified at the Partner Profile level. If this flag is specified, Header and Footer information will be processed prior to and/or following each business document.



Message Header and Footer Information is stored with the Trading Partner Definition.

4. Click the **Run** button.

Process Scheduler Request					
User ID: PTDMO		Run Control ID: HRRpts			
Server Name: [dropdown]		Run Date: 07/14/2000		[BT]	
Recurrence: [dropdown]		Run Time: 1:20:25PM			
Time Zone: [dropdown]		Reset to Current Date/Time			
Process List					
Select	Description	Process Name	Process Type	*Type	*Format
<input checked="" type="checkbox"/>	Outbound EC Agent	ECOUT001	SQR Report	Web	PDF
OK Cancel					

Process Scheduler Request page

5. Specify on which server you want the agent to run.
6. Specify when—and how often—you want the agent to run.

If you're running the EDI Agent once, enter the date and time when you want to run it in the **Run Date** and **Run Time** boxes. They default to the current date and time.

If you're running the agents on a server, the Process Scheduler can run them once or periodically on a specified schedule. You use the **Recurrence** box to specify when and how often to run the database agents.

If you selected a **Run Recurrence** other than **Once**, the recurrence definition sets the run date and time. It overrides any **Run Date/Time** you set.

7. Click the **OK** button to run (or schedule to run) the EDI Agent.

Viewing the EDI Audit Trail

As EDI Agents run, they write status information to tables in your PeopleSoft database. The EDI Manager's **Inquire** menu gives you access to several panels for reviewing the status and the processing history.

Reviewing and Correcting Errors

PeopleSoft delivers a predefined business process named *Manage EDI* which, among other things, routes information about any processing errors it encounters to the EDI Coordinator's worklist. The coordinator can select the item from the worklist to review the error and possibly correct the problem.

To review EDI processing errors

1. From your browser, select PeopleTools, EDI Manager, Inquire, Business Document Errors.

The Business Document Detail panel appears.

This panel gives detailed information about the processing of each line in a PeopleSoft Business Document.

2. Review the processing details for the document.

When you first open the panel, it shows the processing information for the first record in the file, which is a 999 record. Use the outer scroll bar to scroll from one line to the next. The status of each line appears in the **Status** field.

3. Fix the erroneous data in the PeopleSoft Business Document (optional).

This page enables you to edit data from the PeopleSoft Business Document.



Because this panel enables you to change transaction data, you'll want to carefully control access to it. In Operator Security, only give the EDI Coordinator rights to edit this panel.

4. Save the panel group.
5. Select Inquire, Transaction Maintenance.
6. Reset the **EC Queue Status** for the corrected transactions to L.

When the EDI Agent encounters an error while processing a transaction, it sets the status field in the pending transaction queue (ECQUEUE) to E. Resetting the status of a transaction to L makes the Outbound EDI Agent try to process it again.

Packages and Transaction Groups

The EDI transaction set file that you receive over the network from an external trading partner might include any number of transactions bound for any number of internal trading partners. To manage all this the transaction data, the file organizes it into several levels.

- The top level of organization is the *package level*. The package is the entire transaction set file, addressed to your company much as a mail package would be.
- The package can contain one or more *transaction groups*. Each transaction group is a set of

transactions of the same type, with the same trading partners involved.

- Each transaction group includes one or more individual *units of work*. A unit of work is a single transaction that you want to commit or rollback as a whole.

When the (third-party-supplied) EDI translation software creates PeopleSoft Business Documents, it divides up the transaction set file at approximately the transaction group level. A PeopleSoft Business Document typically includes multiple units of work, and may include multiple transaction groups depending on how the translation software is set up.

The PeopleSoft Business Documents do not contain any information about what packages or transaction groups their transactions come from. However, the EDI translation software can provide this information in a separate audit file, and the EDI Agent can copy the information into audit tables in your database.



For the auditing information to be available, the EDI translation software must write the package and transaction group information to an audit file, and the EDI Agent must process this file using the predefined AUDIT inbound map.

To view summary information about a PeopleSoft Business Document

1. Select Inquire, Business Document Summary, Summary.
2. Enter the Business Document ID of the document you want to review.

This pagel displays information about how the EDI Agent processed the specified PeopleSoft Business Document. If the document included multiple transactions, the scroll bar in the lower part of the panel enables you to see the information about each individual transaction.

3. Review the packages and transaction groups that this document is part of (optional).

This page displays information about the EDI packages and transaction groups that the current document's data came from.

To view summary information about an EDI package

1. Select Inquire, Package Log Summary.
2. Enter the Package Control ID for the package whose status you want to review, or press Enter to select from a list of the packages.

This pagel displays the processing information for the selected package. The scroll bars in the lower part of the panel enable you to scroll through the transaction groups that were inside the package and the PeopleSoft Business Documents that were created from the data in the package.

To view summary information about an EDI transaction group

1. Select Inquire, Transaction Group Summary.

2. Enter the Transaction Group Control ID for the group whose status you want to review, or press Enter to select from a list of the groups.

This page displays the processing information for the selected transaction group.

Message Agent



Note. Message Agent is a deprecated integration feature in this PeopleSoft release. We recommend that you *not* develop new integration applications with the Message Agent API's. PeopleSoft's Component Interface is now the recommended method to access PeopleSoft components (panel groups) and the business logic associated with them outside of PeopleSoft on-line pages.



Because the Message Agent has been replaced with component interface, we chose to expend our documentation efforts in the new material. Therefore, in this Message Agent documentation there are many references to message definitions and the message agent server that are no longer valid. The meanings of these terms have changed in the application messaging architecture of Release 8 and beyond. This Message Agent documentation also references the PTDMO database to use for sample message definitions. PTDMO is no longer available to customers or partners. PeopleSoft 8 includes an Integration SDK to replace it.



Warning! If you have compiled C programs that call PeopleSoft 7.5 Message Agent APIs, they must be recompiled with new C headers included with PeopleSoft 8.

Message Agent Overview

To use the Message Agent you must have the following:

- a message definition for Message Agent
 - Message Agent client programs
 - a Message Agent Server
-



Note. This outline follows the same steps an external program would perform to mimic the behavior of the on-line PeopleSoft panels. The specific API calls used in each of these operations are detailed in the Programming the Message Agent section.

Managing Multiple Scroll Levels (Level Mapping)

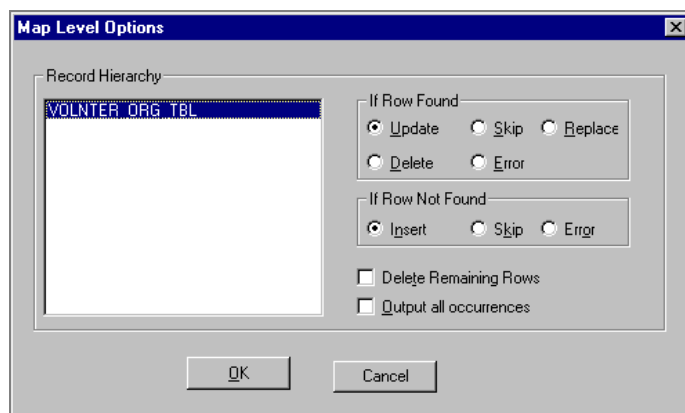
Many PeopleSoft application panels include one or more scroll bars, which enable them to display data from more than one row of data at a time. When you access such a panel using the Message Agent, you have access to multiple rows of data.

The message definition specifies what the Agent does if it finds a row. The Agent actions are to:

- update it
- replace it
- delete it
- skip it
- or error

If the Message Agent does not find a row it will add a new row, skip it or return an error.

As a result of the Level Mapping button in the message definition, panels with multiple rows of data will give you the following:



Map Level Options

If the Message Definition that is being used has the *Output all occurrences* option selected, you also have the option of retrieving data from all the rows inside a level 1 scroll, assuming that the message definition includes their fields as output fields.



This option only works for Level 1 scrolls.

Limitations of Multiple Scroll Levels

Many PeopleSoft application panels include one or more scroll bars, which enable them to display data from more than one row of data at a time. When you access such a panel using the

Message Agent, you have access to multiple rows of data. The Message Agent can only add or update one row of data at a time. So, to add or update multiple rows inside a scroll you will need to make repeated calls to process the message (ProcessMessage).

Adding or Updating Multiple Rows

The Message Agent can only add or update one row of data at a time, except for data inside a level 1 scroll. To add or update multiple rows inside level 2 or level 3 scrolls, you need to process the rows one at a time.

To add or update rows inside a scroll:

1. Specify which message definition to use (StartMessage).
2. If necessary, get the metadata that describes the fields in the message definition (GetFieldList, or the GetFieldInfo method in the OLE interface).
3. Set the values for the first row you want to add or update (SetField). You must set values for the key fields of the main record definition inside the scroll as well as any higher level keys.
4. For level 1 scrolls only, set the values for the next row you want to add or update (SetField). This time, you don't need to specify the level 0 keys again; just provide the new level 1 values. Repeat this step for each row you want to add or update.
5. Process the row (ProcessMessage).
6. Check the results (GetField). You can get the values from the output fields specified in the message definition. If an error occurred, you can request the text of the error message.
7. Repeat steps 3 through 6 for the next record you want to add or update.
8. Disconnect from the Message Agent or start another message.
9. If the message definition has the Delete remaining rows option selected, the Message Agent will remove any rows inside the scroll that you didn't add or update since you started the message at step 1.

Retrieving Multiple Rows

To retrieve the data from an individual row within a scroll, follow the same process as you do to retrieve data from a panel that does not include a scroll bar. You provide the Message Agent with values for all the key fields necessary to identify the row, then use GetField to retrieve the values from the (non-key) data fields specified as output fields in the message definition.

If the message definition you are using has the 'Output all occurrences' option selected, you also have the option of retrieving data from *all* the rows inside a level 1 scroll—assuming, of course, that the message definition includes their fields as output fields.



The Output all occurrences option is available for level 1 scrolls only.

To retrieve data from all rows inside a level 1 scroll:

1. Specify which message definition to use (StartMessage).
2. If necessary, get the metadata that describes the fields in the message definition (GetFieldList, or the GetFieldInfo method in the OLE interface).
3. Set the values for the level 0 key fields, but not for any of the level 1 key fields (SetField). The level 1 keys need to be specified as output fields in the message definition.
4. Process the message (ProcessMessage).
5. Get the results for the first returned row (GetValue). You can get the values from the output fields specified in the message definition.
6. Get the next returned row (FindNextOutputRow), then repeat step 5. Continue until FindNextOutputRow returns the keyword PSMSG_NOTFOUND, which means that you've processed all the returned rows.
7. Disconnect from the Message Agent or start another message.

Message Agent Field Mapping

Which data fields map to which panel fields is designated in the field mapping creation step of the message definition. Field mapping is how the Message Agent knows what values to enter into the correct fields on the PeopleSoft panels.

Programming the Message Agent

The Message Agent APIs provide the means to accomplish many functions with the panel interface. The Message Agent APIs provide you the capability to use the program options:

- Search (field query and file query)
- Scrollbar listing (limited to level 1)
- Prompt and Edit tables
- Search Dialog processing

There are a large number of calls that make up the Message Agent APIs and there are a number of complex issues associated with programming some parts of an external program. This section identifies some of the particular APIs to accomplish the above programming options and provides a number of very simple program examples that will detail each area of programming the Message Agent APIs.

Message Agent API Setup

The data transferred to a message or panel interface is based on the basic elements of the Message Agent API programming. The Message Agent API section outlines in detail each API function and operation. This section identifies the particular APIs to accomplish the above programming options.

Basic program setup

A basic program using the Message Agent APIs may contain only a few APIs. The following APIs can be used to create a basic Message/Panel Interface program.

To create a basic Message/Panel Interface program

1. Use the following APIs:

Connect - Connects to a specific application server machine/port and log on.

StartMessage - specify the message definition to use.

SetField – Set field calls to the set key fields and set field for data entry (if you want to add/update or correct).

ProcessMessage - enter the data into the panel.

Disconnect - disconnect from the application server.

Optionally, if retrieving data is required you would include:

FindFirstField

GetValue

FindNextField

FindNextOutputRow

The above programming completes the following:

- Ensures the program can connect to a Message Agent.
- Identifies what message definition to use.
- Sets the value of a message field to be mapped into a panel buffer when the message is processed.
- Triggers message processing after all input message fields have been sent.
- Any fields mapped as output will be returned if an unique name is identified.

Examples of simple basic API program are:

- Add a level 0 row
- Add or update row (**non-level 0**)
- Read a row
- Add or update multiple rows (**non-level 0**)
- Read multiple rows
- C/C++ program

Sample API programs are listed in the Message Agent Examples section of this documentation.

Searching for Records/Search Dialog Processing

You may want to retrieve information from records. Or you may want to add or update a record in the PeopleSoft database. To do a record search you must first specify the key values for the record. If you are retrieving an existing on-line, you have two options for specifying the key values:

- Enter complete values for each of the key fields
- Enter partial values (search criteria) in the key fields, then select from a list of the records matching the partial values

When you enter partial values in the online system, it displays a list box with the matching records. The Message Agent API includes functions that mimic this operation. You can create a Message Agent API program to perform search and retrieve functionality using the following APIs:

To create a search API program

1. Use the following APIs:

StartMessage

GetSearchFieldInfo

FindField

GetFieldCount

GetFieldList

The above field search APIs query for field information. If you require a query of file information you can include the following APIs:

GetFieldNameLength

GetFieldName

ProcessSearchDialog

GetSearchFieldCount
GetSearchList
GetSearchRecordLength
GetSearchRecord
FindFirstListboxRow
FindFirstListBoxField
GetListBoxRow
FindNextListBoxField
FindNextLastBoxRow
FindListBoxRow
FindListboxField
GetListBoxRowCount
ProcessMessage

You may use the following two APIs to query information about data:

GetMaxValueLength
GetValueLength

The above programming completes the following:

- Specifies which message definition to use (StartMessage).
- Get the name of the search record (GetSearchRecord) and its field attribute information (GetSearchList or GetSearchFieldInfo).
- Enter search values in one or more fields (SetField).
- Process the search dialog box (ProcessSearchDialog).
- Find out how many records matched the search criteria (GetListBoxRowCount).
- Select the record you want. You can scroll through the records using FindFirstListBoxRow and FindNextListBoxRow, or go to a specific row using FindListBoxRow. You get the values from a row using GetListBoxRow or GetListBoxFieldInfo.
- Use the values from the selected record to set the key values for the message (SetField).
- Process the message (ProcessMessage).



Note. Typical use of the Search Dialog Processing APIs is with Windows clients to emulate PeopleSoft panels.

Examples of search API programs are:

- Get message definition field information
- Get search dialog information
- Do search dialog processing

Sample API programs are listed in the Message Agent Examples section of this documentation.

Edit Table Processing

Some record fields have an *edit table* associated with them. The edit table lists valid values for the field. For example, a State field may have an associated edit table that lists the valid state codes. When you enter data, the system checks it against the edit table; if you enter a value that does not appear on the table, you get an error.

Edit tables are also called *prompt tables*, because in the online system, a user can press F4 to display the contents of the table and select a value. You can mimic this behavior using a Message Agent API program.

To create an edit/prompt table API program

1. Use the following APIs:

StartMessage

GetFieldList

FindFirstPromptValueRow

FindNextPromptValueRow

FindPromptValueRow

GetPromptValueRowCount

GetPromptValueRow

GetPromptValueFieldCount

GetEditTableFieldList

ProcessMessage

Using the above programming APIs, the following events are completed:

- Gets the attributes of the field (GetFieldList, GetFieldInfo, GetSearchList, or GetSearchInfo). If the field has an edit table, it appears as the last attribute in the field attribute structure.
- Gets the attributes for the fields in the edit table (GetEditTableFieldList or GetEditTableFieldInfo).
- Process the edit table request (ProcessPromptTable).
- Select the edit table record you want. You can find out how many records matched the search criteria using GetPromptValueRowCount. You can scroll through the records using FindFirstPromptValueRow and FindNextPromptValueRow, or go to a specific row using FindPromptValueRow. You get the value from a row using GetPromptValueRow.

Examples of search API programs are:

- Get edit/prompt table information
- Do edit/prompt table processing
- Do search dialog processing

Sample API programs are listed in the Message Agent Examples section of this documentation.

Message Agent Examples

Add a level 0 row

This example shows how you would add a level 0 row using the Message Agent. The example uses the AddL0 message definition in the MSGAGT_EXAMPLES business process. The message definition references the VOLUNTEER_ORG_TABL panel in “add” mode. It maps 4 fields, all for input: VOLUNTEER_ORG, EFFDT, EFFSTATUS, and DESCR.

After running this program you can use the online panels to look at the level 0 row you just entered.

The flow of the program is as follows:

Connect - connect to a specific (prompted for) app server machine/port and logon.

StartMessage - specify the message definition to use.

SetField - specify the level 0 keys, and the level 0 data.

ProcessMessage - enter the data into the panel.

Disconnect - disconnect from the app server machine.

Public Sub Main()

.....

```

'
' Simple Message Agent program to add a L0 row.
'
'
'
' Connect to the Message Agent server.
ServerPort = InputBox("Please input server machine:port")
Result = Mag.Connect(ServerPort, "PTDMO", "PTDMO")
If Result <> PSMSG_OK Then PrintError ("Error in Connect")

' Identify what message definition to use.
Result = Mag.StartMessage("MsgAgtExamples", "AddL0", False)
If Result <> PSMSG_OK Then PrintError ("Error in StartMessage")

' Set the key fields.
Result = Mag.SetField("VOLUNTEER_ORG", "RPS")
If Result <> PSMSG_OK Then PrintError ("Error in SetField")
Result = Mag.SetField("EFFDT", "01/01/98")
If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Set the data field.
Result = Mag.SetField("DESCR", "Retired programmer society")
If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Process the message
Result = Mag.ProcessMessage(nReplyOption)
If Result <> PSMSG_OK Then PrintError ("Error in ProcessMessage")

' All done disconnect from the Message Agent server.

```



```
Result = Mag.Disconnect()
```

```
MsgBox "Success"
```

```
End Sub
```

Add or update a row (non-level 0)

This example shows how you would either add or update a non level 0 row. This example uses the AddUpdRow message definition in the MSGAGT_EXAMPLES business process. This message definition references the NAMES panel in “update/display” mode. It maps 4 fields, all for input: EMPLID, NAME_TYPE, NAME_PART, and PREFERRED_NAME.

After running this program you can use the online panels to look at the level 1 row you just entered.

The flow of the program is as follows:

Connect - connect to a specific (prompted for) app server machine/port and logon.

StartMessage - specify the message definition to use.

SetField - specify the level 0 keys, and the level 1 data to insert/update.

ProcessMessage - enter the data into the panel.

Disconnect - disconnect from the app server machine.

```
Public Sub Main()
```

```
.....
```

```
'
```

```
' Simple Message Agent program to add/upd a non-L0 row.
```

```
'
```

```
.....
```

```
' Connect to the Message Agent server.
```

```
ServerPort = InputBox("Please input server machine:port")
```

```
Result = Mag.Connect(ServerPort, "PTDMO", "PTDMO")
```

```
If Result <> PSMSG_OK Then PrintError ("Error in Connect")
```

' Identify what message definition to use.

Result = Mag.StartMessage("MsgAgtExamples", "AddUpdRow", False)

If Result <> PSMSG_OK Then PrintError ("Error in StartMessage")

' Set the key fields.

Result = Mag.SetField("EMPLID", "8001")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

Result = Mag.SetField("NAME_TYPE", "OTH")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

Result = Mag.SetField("NAME_PART", "F")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Set the data field.

Result = Mag.SetField("PREFERRED_NAME", "1")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Process the message

Result = Mag.ProcessMessage(nReplyOption)

If Result <> PSMSG_OK Then PrintError ("Error in ProcessMessage")

' All done disconnect from the Message Agent server.

Result = Mag.Disconnect()

MsgBox "Success"

End Sub

Read a row

This example shows how you would read a row using the Message Agent. The example uses the ReadRow message definition in the MSGAGT_EXAMPLES business process. The message definition references the NAMES panel and record, and runs in “update/display” mode. It maps 4 fields, the first for input (the key) and the others for output: EMPLID, NAME_TYPE, NAME_PART, and PREFERRED_NAME.

The flow of the program is as follows:

Connect - connect to a specific (prompted for) app server machine/port and logon.

StartMessage - specify the message definition to use.

SetField - specify the level 0 key.

ProcessMessage - perform the operation.

FindFirstField - get the first field in the row.

GetValue - get the field value.

FindNextField - get the next field (and loop), if one.

Disconnect - disconnect from the app server machine.

Public Sub Main()

```
'
' Example Message Agent program to read a row.
```

Dim Value As String

' Connect to the Message Agent server.

```
ServerPort = InputBox("Please input server machine:port")
```

```
Result = Mag.Connect(ServerPort, "PTDMO", "PTDMO")
```

```
If Result <> PSMSG_OK Then PrintError ("Error in Connect")
```

```
' Identify what message definition to use.

Result = Mag.StartMessage("MsgAgtExamples", "ReadRow", False)
If Result <> PSMSG_OK Then PrintError ("Error in StartMessage")

' Set the key field.
Result = Mag.SetField("EMPLID", "8001")
If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Process the message
Result = Mag.ProcessMessage(nReplyOption)
If Result <> PSMSG_OK Then PrintError ("Error in ProcessMessage")

' Get all the fields in the output row.
Result = Mag.FindFirstField()
If Result <> PSMSG_OK Then PrintError ("Error in FindFirstField")
Do
    Result = Mag.GetValue(Value, 30)
    If Result <> PSMSG_OK Then PrintError ("Error in GetValue")
    MsgBox Value
    Result = Mag.FindNextField()
Loop Until Result <> PSMSG_OK

' Done
MsgBox "Operation completed successfully"

' All done disconnect from the Message Agent server.
Disconnect:
Result = Mag.Disconnect()
```

End Sub

Add or update multiple rows (non-level 0)

This example shows how you would add or update multiple non-level 0 rows. The example uses the AddUpdRows message definition in the MSGAGT_EXAMPLES business process. The message definition references the NAMES panel and record, and runs in “update/display” mode. It maps 4 fields for input EMPLID, NAME_TYPE, NAME_PART, and PREFERRED_NAME.

After running this program you can use the online panels to look at the rows you just entered.

The flow of the program is as follows:

Connect - connect to a specific (prompted for) app server machine/port and logon.

StartMessage - specify the message definition to use.

SetField - specify the keys and data for all the rows.

ProcessMessage - perform the operation.

Disconnect - disconnect from the app server machine.

Public Sub Main()

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'
' Simple Message Agent program to add/upd multiple rows.
'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

' Connect to the Message Agent server.
ServerPort = InputBox("Please input server machine:port")
Result = Mag.Connect(ServerPort, "PTDMO", "PTDMO")
If Result <> PSMSG_OK Then PrintError ("Error in Connect")

' Identify what message definition to use.
Result = Mag.StartMessage("MsgAgtExamples", "AddUpdRow", False)
If Result <> PSMSG_OK Then PrintError ("Error in StartMessage")

```

' Set the L0 key.

Result = Mag.SetField("EMPLID", "8001")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Set the data fields for the first row.

Result = Mag.SetField("NAME_TYPE", "OTH")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

Result = Mag.SetField("NAME_PART", "F")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

Result = Mag.SetField("PREFERRED_NAME", "2")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Set the data fields for the second row.

Result = Mag.SetField("NAME_TYPE", "OTH")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

Result = Mag.SetField("NAME_PART", "L")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

Result = Mag.SetField("PREFERRED_NAME", "3")

If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Process the message

Result = Mag.ProcessMessage(nReplyOption)

If Result <> PSMSG_OK Then PrintError ("Error in ProcessMessage")

' All done disconnect from the Message Agent server.

Result = Mag.Disconnect()

MsgBox "Success"

End Sub

Read multiple rows

This example shows how you would add or update multiple non-level 0 rows. The example uses the AddUpdRows message definition in the MSGAGT_EXAMPLES business process. The message definition references the NAMES panel and record, and runs in “update/display” mode. It maps 4 fields for input EMPLID, NAME_TYPE, NAME_PART, and PREFERRED_NAME.

After running this program you can use the online panels to look at the rows you just entered.

The flow of the program is as follows:

Connect - connect to a specific (prompted for) app server machine/port and logon.

StartMessage - specify the message definition to use.

SetField - specify the level 0 keys, and the level 1 data to insert/update.

ProcessMessage - perform the operation.

FindFirstField - point to the first field in the row.

GetValue - get the value for this field.

FindNextField - point to the next field (loop for all fields in the row).

FindNextOutputRow - point to the next output row (loop for all rows).

Disconnect - disconnect from the app server machine.

Public Sub Main()

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'
' Example Message Agent program to read multiple rows.
'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Dim Value As String

' Connect to the Message Agent server.

ServerPort = InputBox("Please input server machine:port")

```
Result = Mag.Connect(ServerPort, "PTDMO", "PTDMO")

If Result <> PSMSG_OK Then PrintError ("Error in Connect")

' Identify what message definition to use.
Result = Mag.StartMessage("MsgAgtExamples", "ReadRows", False)
If Result <> PSMSG_OK Then PrintError ("Error in StartMessage")

' Set the key fields.
Result = Mag.SetField("EMPLID", "8001")
If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Process the message
Result = Mag.ProcessMessage(nReplyOption)
If Result <> PSMSG_OK Then PrintError ("Error in ProcessMessage")

' Get all the output rows.
Do
    ' Get all the fields in the output row.
    Result = Mag.FindFirstField()
    If Result <> PSMSG_OK Then PrintError ("Error in FindFirstField")
    Do
        Result = Mag.GetValue(Value, 30)
        If Result <> PSMSG_OK Then PrintError ("Error in GetValue")
        MsgBox Value
        Result = Mag.FindNextField()
    Loop Until Result <> PSMSG_OK

Result = Mag.FindNextOutputRow()
Loop Until Result <> PSMSG_OK
```



```
' Done
```

```
MsgBox "Operation completed successfully"
```

```
' All done disconnect from the Message Agent server.
```

```
Disconnect:
```

```
Result = Mag.Disconnect()
```

```
End Sub
```

Get error information

This example shows how you would get error information from the message agent. The example uses the AddUpdRows message definition in the MSGAGT_EXAMPLES business process. The message definition references the NAMES panel and record, and runs in “update/display” mode. It maps 4 fields for input EMPLID, NAME_TYPE, NAME_PART, and PREFERRED_NAME. The example will attempt to input an invalid value to the NAME_PART field, which will result in an error in the process message.

Note that other Message Agent API calls can result in errors that create error text/explain text, however the error record/fields are not valid.

The flow of the program is as follows:

Connect - connect to a specific (prompted for) app server machine/port and logon.

StartMessage - specify the message definition to use.

SetField - specify the keys with one setting an invalid value.

ProcessMessage - perform the operation.

GetErrorRecordName - get name of record that got the error.

GetErrorFieldName - get name of field that got the error.

GetErrorText - get the error text associated with the last error.

GetErrorExplainText - get the explain text associated with the last error.

Disconnect - disconnect from the app server machine.

```
Public Sub Main()
```

```
Dim Str As String
```

```
.....
```

```
,
```

```
' Basic Visual Basic Message Agent program.
```

```
,
```

```
.....
```

```
' Connect to the Message Agent server.
```

```
ServerPort = InputBox("Please input server machine:port")
```

```
Result = Mag.Connect(ServerPort, "PTDMO", "PTDMO")
```

```
If Result <> PSMSG_OK Then PrintError ("Error in Connect")
```

```
' Identify what message definition to use.
```

```
Result = Mag.StartMessage("MsgAgtExamples", "AddUpdRow", False)
```

```
If Result <> 0 Then PrintError ("Error in StartMessage")
```

```
' Set the key fields, with the last one being bad.
```

```
Result = Mag.SetField("EMPLID", "8001")
```

```
If Result <> PSMSG_OK Then PrintError ("Error in SetField")
```

```
Result = Mag.SetField("NAME_TYPE", "OTH")
```

```
If Result <> PSMSG_OK Then PrintError ("Error in SetField")
```

```
Result = Mag.SetField("NAME_PART", "X")
```

```
If Result <> PSMSG_OK Then PrintError ("Error in SetField")
```

```
' Process the message
```

```
Result = Mag.ProcessMessage(nReplyOption)
```

```
If Result = 0 Then PrintError ("Error expected in StartMessage")
```

' We expect an error here

Result = Mag.GetErrorRecordName(Str, 50)

If Result <> 0 Then PrintError ("Error in GetErrorRecordName")

Result = Mag.GetErrorFieldName(Str, 50)

If Result <> 0 Then PrintError ("Error in GetErrorFieldName")

Result = Mag.GetErrorText(Str, 50)

If Result <> 0 Then PrintError ("Error in GetErrorText")

Result = Mag.GetErrorExplainText(Str, 100)

If Result <> 0 Then PrintError ("Error in GetErrorExplainText")

' Done

MsgBox "Operation completed successfully"

' All done disconnect from the Message Agent server.

Disconnect:

Result = Mag.Disconnect()

End Sub

Get message definition field information

This example shows how you would get the field information in the message definition using the Message Agent. The example uses the AddUpdRow message definition in the MSGAGT_EXAMPLES business process.

The flow of the program is as follows:

Connect - connect to a specific (prompted for) app server machine and logon.

StartMessage - specify the message definition to use.

GetFieldCount - get the number of fields in the message definition.

GetFieldInfo - get field information for a specific field (loop for all fields).

Public Sub Main()

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'
' Example Message Agent program to get message definition field info.
'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Dim i&

Dim FieldInfo As Object

```

' Connect to the Message Agent server.
ServerPort = InputBox("Please input server machine:port")
Result = Mag.Connect(ServerPort, "PTDMO", "PTDMO")
If Result <> PSMSG_OK Then PrintError ("Error in Connect")

' Identify what message definition to use.
Result = Mag.StartMessage("MsgAgtExamples", "AddUpdRow", False)
If Result <> 0 Then PrintError ("Error in StartMessage")

' Process the number of fields in the message definition.
Result = Mag.GetFieldCount(i)
If Result <> 0 Then PrintError ("Error in GetFieldCount")

' Loop through all the fields.
For j = 0 To i - 1

```

```

' Get the information about this field.

Result = Mag.GetFieldInfo(j, FieldInfo)

If Result <> 0 Then PrintError ("Error in GetFieldInfo")

FieldName = FieldInfo.szFieldName

FieldType = FieldInfo.fFieldType

FieldSize = FieldInfo.nFieldSize

FieldMapUse = FieldInfo.fFieldMapUse

FieldMapTransfer = FieldInfo.fFieldMapTransfer

FieldLevel = FieldInfo.nFieldLevel

Next j

' Done

MsgBox "Operation completed successfully"

' All done disconnect from the Message Agent server.

Disconnect:

Result = Mag.Disconnect()

End Sub

```

Get search dialog information

You can get information about all of the search record fields using the message agent. This procedure is exactly like getting the information about the fields in a message definition, except the Message Agent calls have different names (see Get message definition field information).

GetSearchFieldCount will get the number of fields in the message definitions search record definition. GetSearchFieldInfo is used to get information about a specific field.

You can also get the name of the search record for the current message definition using the GetSearchRecord call, and the length of this record name using the GetSearchRecordLength.

Note that if you are using C/C++ PSMsgGetSearchList is the call you should use, and it returns all the information about the fields. You must pass it a pointer to an array big enough for all of the fields.

Do search dialog processing

This example shows how you would do search dialog processing using the Message Agent. The example uses the ReadRow message definition in the MSGAGT_EXAMPLES business process.

The flow of the program is as follows:

Connect - connect to a specific (prompted for) app server machine and logon.

StartMessage - specify the message definition to use.

SetField - set the key value for search processing.

ProcessSearchDialog - do the search processing.

FindFirstListBoxRow - point to the first row.

GetListBoxFieldCount - get the count of fields in the row.

GetListBoxFieldInfo - get information on each field in the row.

FindNextListBoxRow - get the next row (loop for all rows).

Public Sub Main()

```
'
' Example Message Agent program to do search dialog processing.
'
```

Dim i&

Dim RowInfo As Object

' Connect to the Message Agent server.

```
ServerPort = InputBox("Please input server machine:port")
```

```
Result = Mag.Connect(ServerPort, "PTDMO", "PTDMO")
```

```
If Result <> PSMSG_OK Then PrintError ("Error in Connect")
```

' Identify what message definition to use.

```
Result = Mag.StartMessage("MsgAgtExamples", "ReadRow", False)
If Result <> PSMSG_OK Then PrintError ("Error in StartMessage")

' Set the key field.
Result = Mag.SetField("EMPLID", "80")
If Result <> PSMSG_OK Then PrintError ("Error in SetField")

' Process the message
Result = Mag.ProcessSearchDialog(0)
If Result <> PSMSG_OK Then PrintError ("Error in ProcessSearchDialog")

' Get all the rows in the search results.
Result = Mag.FindFirstListBoxRow()
If Result <> PSMSG_OK Then PrintError ("Error in FindFirstListBoxRow")
Do
    ' Get all of the fields in the row.
    Result = Mag.GetListBoxFieldCount(i)
    If Result <> PSMSG_OK Then PrintError ("Error in GetListBoxFieldCount")
    For j = 0 To i - 1
        Result = Mag.GetListBoxFieldInfo(j, RowInfo)
        If Result <> PSMSG_OK Then PrintError ("Error in GetListBoxFieldInfo")
        MsgBox RowInfo.szValue
    Next j
    Result = Mag.FindNextListBoxRow()
Loop Until Result <> PSMSG_OK

' Done
MsgBox "Operation completed successfully"
```

' All done disconnect from the Message Agent server.

Disconnect:

Result = Mag.Disconnect()

End Sub

Get edit/prompt table information

You can get information about all of the information about edit/prompt table fields using the message agent. This procedure is exactly like getting the information about the fields in a message definition, except the Message Agent calls have different names (see Get message definition field information).

GetEditTableFieldCount will get the number of fields in the prompt/edit table (record).
GetEditTableFieldInfo is used to get information about a specific field in the prompt/edit table (record).

Note that if you are using C/C++ PSMsgEditTableFieldList is the call you should use, and it returns all the information about the fields. You must pass it a pointer to an array big enough for all of the fields.

Do edit/prompt table processing

This example shows how you would get the field information in the message definition using the Message Agent.

The flow of the program is as follows:

Connect - connect to a specific (prompted for) app server machine and logon.

StartMessage - specify the message definition to use.

ProcessPromptTable - get the edit/prompt table.

FindFirstPromptValueRow - point to the first row

GetPromptValueFieldCount - get the number of fields in the message definition.

GetPromptValueInfo - get field information for a specific field (loop for all fields).

FindNextPromptValueRow - point to the next row (loop for all rows).

Public Sub Main()

.....


```

'
' Example Message Agent program to do edit/prompt table processing.
'
'
' =====

Dim i&
Dim RowObj As Object

' Connect to the Message Agent server.
ServerPort = InputBox("Please input server machine:port")
Result = Mag.Connect(ServerPort, "PTDMO", "PTDMO")
If Result <> PSMSG_OK Then PrintError ("Error in Connect")

' Identify what message definition to use.
Result = Mag.StartMessage("MsgAgtExamples", "ReadRow", False)
If Result <> PSMSG_OK Then PrintError ("Error in StartMessage")

' Process the message
Result = Mag.ProcessPromptTable("SP_BUIN_NONVW", 0)
If Result <> PSMSG_OK Then PrintError ("Error in ProcessPromptTable")

' Get all the rows in the search results.
Result = Mag.FindFirstPromptValueRow()
If Result <> PSMSG_OK Then PrintError ("Error in FindFirstPromptValueRow")
Do
    ' Get all of the fields in the row.
    Result = Mag.GetPromptValueFieldCount(i)
    If Result <> PSMSG_OK Then PrintError ("Error in GetEditTableFieldCount")
    For j = 0 To i - 1

```

```

    Result = Mag.GetPromptValueInfo(j, RowObj)

    If Result <> PSMSG_OK Then PrintError ("Error in GetPromptValueInfo")

    MsgBox RowObj.szValue

Next j

Result = Mag.FindNextPromptValueRow()

Loop Until Result <> PSMSG_OK

' Done

MsgBox "Operation completed successfully"

' All done disconnect from the Message Agent server.

Disconnect:

Result = Mag.Disconnect()

End Sub

```

C/C++ program

This example shows a very simple Message Agent program written in C/C++. It performs the same functionality as “Add or update a row (non-level 0) Visual Basic example. This message definition references the NAMES panel in “update/display” mode. It maps 4 fields, all for input: EMPLID, NAME_TYPE, NAME_PART, and PREFERRED_NAME.

Note that all Message Agent calls have the “PSMsg” string at the beginning and the PSMsgConnect sets a “context” that is used by subsequent calls.

After running this program you can use the online panels to look at the level 1 row you just entered.

The flow of the program is as follows:

PSMsgConnect - connect to a specific (prompted for) app server machine/port and logon.

PSMsgStartMessage - specify the message definition to use.

PSMsgSetField - specify the level 0 keys, and the level 1 data to insert/update.

PSMsgProcessMessage - perform the operation.

PSMsgDisconnect - disconnect from the app server machine.

```
#include <string.h>
```

```
#include <stdio.h>
```

```
#include <src/inc/psmsgapi.h>
```

```
void PrintError(char *);
```

```
PSMSGHANDLE    hCtx;
```

```
main()
```

```
{
```

```
PSOPERATOR    Opr;
```

```
int            Result;
```

```
int            nReplyOption;
```

```
strcpy(Opr.szOprId, "PTDMO");
```

```
strcpy(Opr.szOprPswd, "PTDMO");
```

```
Result = PSMsgConnect(PSMSG_APIVERSION, "207.135.60.205:7000", &Opr, &hCtx);
```

```
if (Result != PSMSG_OK)
```

```
    PrintError("Error in Connect");
```

```
Result = PSMsgStartMessage(hCtx, "MSGAGT_EXAMPLES", "AddUpdRow", 0);
```

```
if (Result != PSMSG_OK)
```

```
    PrintError("Error in StartMessage");
```

```
Result = PSMsgSetField(hCtx, "EMPLID", "8001");
```

```
if (Result != PSMSG_OK)
```

```
    PrintError("Error in SetField");

    Result = PSMsgSetField(hCtx, "NAME_TYPE", "OTH");
    if (Result != PSMSG_OK)
        PrintError("Error in SetField");

    Result = PSMsgSetField(hCtx, "NAME_PART", "F");
    if (Result != PSMSG_OK)
        PrintError("Error in SetField");

    Result = PSMsgSetField(hCtx, "PREFERRED_NAME", "me");
    if (Result != PSMSG_OK)
        PrintError("Error in SetField");

    Result = PSMsgProcessMessage(hCtx, &nReplyOption);
    if (Result != PSMSG_OK)
        PrintError("Error in ProcessMessage");

    Result = PSMsgDisconnect(hCtx);
    if (Result != PSMSG_OK)
        PrintError("Error in Disconnect");

    printf("Success\n");

    return(0);

}
```

```
/* Print an error and die */  
  
void      PrintError(char *pStr)  
{  
  
    printf("%s\n", pStr);  
    PSMsgDisconnect(hCtx);  
  
    //exit(1);  
  
}
```

Troubleshooting the Message Agent

The following section provides a means to correct possible Message Agent programming problems.

Message Agent Debugging

You just attempted processing a Message Agent transaction but it fails with an error. In the beginning, you may see errors frequently but don't get frustrated. In time, you'll master how to create a Message Agent transaction from start to finish, from creating PeopleSoft data objects in Application Designer to placing your finishing touches in your program. This section will provide you with tips and guidelines on how to troubleshoot problem areas along the way.

Let's start from the beginning. What should your initial steps be to locate where the problem lies?

Installation

As of PeopleTools 7.5, the client is required only to have a subset of PeopleSoft dll's to have access to the Message Agent and invoke Message Agent API functions. After downloading client installation files from the PeopleSoft CD to a "file server," you can install only the necessary client dll's on Windows by running a batch program located under the SETUP directory. For UNIX clients, you must perform a server transfer of files and then run a shell script to copy the appropriate files to the UNIX client.



For more information and details about installation setup, please refer to the Installation Documentation.

Administration

You can have multiple instances of the Message Agent server (PSAPISRV) running for a given Application Server. In addition, you have the option, like other Application Server servers, to recycle the Message Agent server after a certain number of requests were performed. This option may be performed if for some reason the Message Agent servers crashed. The PeopleTools 7.5 Message Agent architecture leverages off the capabilities of our Tuxedo servers. Tuxedo is responsible for queue management of requests from clients and management of servers. Tuxedo also provides a monitoring capability. Please review our Administration Documentation for further details.

Declaring functions

Some difficulties may arise from simply calling the functions from a program. For Visual Basic programmers, we recommend using the OLE Automation Server API's. You can create an OLE Automation Server object two ways. It's rather easy to program once you get the hang of it. The two programming optionals are:

Programming an OLE Automation Server in Visual Basic

To Program a OLE Automation Server

1. Create a reference to the type library psmas.tlb.
2. Declare a variable of type CMagAutoServer in your program. From here, you have access to its member functions like Connect, StartMessage, ProcessMessage and Disconnect.

Alternative programming option of a OLE Automation Server

3. Declare a variable as a generic object
4. Set the variable to what's returned by CreateObject ("PeopleTools.MessageAgent").

Both options accomplish the same results but performance is quicker using the first option. You can also call the C API's from Visual Basic but you must declare and create aliases for these functions. Refer to the examples included in the Message Agent documentation for further details.

For C/C++ programmers, just include psmsgapi.h in your programs and you'll have access to all functions and global definitions declared there. Note that the header file is compliant to both C and C++ compilers now. Or, you can dynamically load psmsg.dll (LoadLibrary in Windows programming) and obtain function pointers to the API's you need.

For Java programmers, you will have to create wrapper code around the C API's.



Your program can invoke Message Agent functions from either a Windows-client or UNIX-client, unlike previous versions.

Connecting

Great, now you're trying to connect to the Application Server. Make sure you have the proper machine name/IP address and port. There is really no way to query your network for all active Application Servers. You must know this ahead of time. The valid syntax for address and port connection are:

```
//machine name:port,
```

```
//ip address:port,
```

where:

the machine and IP address are the network identifiers for the machine

and the port is the numbered identifier for the Tuxedo domain of the connecting machines.

Remember, the PeopleTools 7.5 Message Agent only supports one active connection per process. That is, you cannot call the Connect function twice without calling the Disconnect function in between them. This is a result of a Tuxedo limitation. The Message Agent does not support the round-robin approach that our Windows client can be configured to do.

StartMessage

If you have gotten passed connecting to the Application Server and now StartMessage is returning an error, most likely, you have specified the wrong or invalid parameters in this function call. Verify that you are accessing the correct Activity and Message Definition. Make sure you are identifying these objects by their name and not by their label/description.

ProcessMessage

What do you do if your ProcessMessage function call returns PSMSG_ERROR. Let's first concentrate on the functions that retrieve error information. For the most part, they'll provide you with enough information to figure out what went wrong. Refer to the Message Agent examples to get an idea on how to use them.

PSMsgGetErrorText

You may notice messages returned from PSMsgGetErrorText like *"Row exists in ADD or DATAENTRY mode"* or *"More than 1 row exists for search keys"*. In each case, you have not provided enough key values to access a particular mode for a "panel group." You must ensure that the key value you provide is a unique identifier to a row. Take a look at what search record you have defined to access a particular panel group. Are you supplying enough key values to the search record for it to perform its duty? Please note that the Message Agent can not default values into the search record like that of online processing. You must manually set those values through the Message Agent. For example, say you are hiring a new employee online. When you perform this action, a default of NEW can be seen in the search dialog for Employee ID. The user doesn't have to type any values in but can just hit return to accept the default. However, the Message Agent is unaware of this. So, the user has to map "NEW" to the search field for Employee ID through the Message Agent using PSMsgSetField. Currently, the Message Agent

does not simulate online search dialog processing to its fullest (e.g., no Search* PeopleCode is invoked).

SQL and PeopleCode Tracing

An important part of debugging Message Agent transactions is learning how to read and interpret trace files. From it, you can figure out what may have went wrong during the transaction. A major change to the PeopleTools 7.5 Message Agent is that traces reside on the Application Server instead of the client. You should coordinate with your administrator on obtaining access to these log files. There are two kinds of tracing, SQL and PeopleCode. There are several ways to invoke both of them. Two options are:

- **Trace all SQL and PeopleCode** - You can trace all SQL and PeopleCode performed by the Application Server by configuring the TraceSQL and TracePC flags on the Application Server. As one can surmise, the trace logs can get quite large over time since the Application Server must trace every SQL transaction and every PeopleCode program it executed.
- **Trace SQL and PeopleCode generated by specific clients** - You can trace SQL and PeopleCode generated by specific clients. You set the TraceSQLMask and TracePCMask parameters on the Application Server and set the appropriate trace flags on the client via Configuration Manager. Refer to the Administration documentation for more details.

Here are some common messages that appear in your trace file when the flag for Message Agent Information is turned on.

Message Set Numbers and Message Numbers

- **Bad map level option** - For records positioned at Level 1, 2 or 3, the Message Agent will retrieve level mapping information when accessing data at those levels. This information is defined in your Level Mapping options in your Message Definition. Sometimes, this message will appear if your panel layout is not structured properly.
- **Record field not found** - The Message Agent attempted to locate a record.field reference on that panel group but could not find it. Verify that your Message Definition field mapping is pointing to the correct record.field and that the panel group contains that record.
- **Unable to update value** - During the course of the transaction, the value you inserted for a particular field could not be properly saved. Most likely, the format of the data is invalid. In other cases, it could be that PeopleCode for that particular field did not properly execute.
- **Record not found in current panel context** - The Message Agent could not locate the record within the panel group. Verify that the record exists by reviewing the panels' layout.
- **Unknown form field** - The field mapping you used is not valid. Note that field mappings are case sensitive.

“MapCore:” statements

- **Invalid data format** - Pretty much self-explanatory. If you are inserting any date fields, verify they are in the proper format for that client.

- **Could not find record field** - The Message Agent tried to retrieve the record and field it needs to map data to but could not. Verify the record and field exist in the database and panel.
- **Unable to find Recfield** - The Message Agent had difficulties locating a record or field in your field mapping.
- **Unable to find entry for this map level** - The Message Agent tried to locate the record at a specific scroll level but could not find it. Make sure you're mapping to the appropriate record at a particular level. The indentation in the field mapping gives you a visual of what records lie at what scroll level. Review your panel layout.
- **Skipping parent key** - Usually of no concern. It just indicates that the Message Agent was traversing through child records and looking for keys to identify a particular row of data. It will inherit key values from parent records.

If you were able to perform the transaction successfully with PeopleCode out of the picture, begin adding back programs. It could be that the PeopleCode you wrote is not executing properly.



Since the Message Agent runs on the application server, all PeopleCode is executed on the application server. You may want to run the Validation Wizard with the web option turned on to verify that all your PeopleCode can be executed on the application server. Review values being used by PeopleCode functions.

Tips for Message Agent

Inserting multiple Level 1 rows.

You have the ability to insert multiple Level 1 rows of data with one ProcessMessage call through the Message Agent. You cannot use this same procedure to insert multiple Level 0, 2 or 3 rows of data in one transaction. The idea is to set all Level 0 key and non-key fields and then loop through all Level 1 key and non-key fields. Once that is completed, issue a ProcessMessage to save that data. Refer to the Message Agent examples in the documentation for more details.

Retrieving multiple output rows.

Again, this is limited to Level 1 scrolls. Make sure the level 1 record has the flag "Output All Occurrences" checked on. Refer to the Message Agent examples for further details on navigating through output lists.

Using %MessageAgent in PeopleCode

You can isolate what PeopleCode functions are being executed by encapsulating those lines of code with if-then statements using %MessageAgent as a criteria. The PeopleCode variable %MessageAgent will contain an actual value when the Message Agent is involved in the transaction.

Duplicate Keys

Records containing duplicate keys are not supported by the Message Agent. In my opinion, the notion of duplicate keys is bad RDBMS syntax. The only workaround is to add another key to the record so a unique set of key values can identify each and every row. You must remove the Duplicate Key field attribute from the record.

“Specified record already exists – update?”

When in Add mode, you may accidentally insert key values for a row that already exist in the database. The above message will appear and gives you the option to update the record or do nothing. Since you can't drag your mouse pointer over a push-button to answer this dialog through the Message Agent, you may wonder what you can do. One of the API's called PSMsgSetOptions allows you to answer this message ahead of time. You can pass in a parameter with a value of PSMSG_OPT_CHANGEMODEADDUPDATE to answer yes to the above dialog. If this option is not set, the Message Agent will assume the answer is no. You should set this option before you call ProcessMessage. This option will be set for the life span of your connection to the Application Server until you call the disconnect function or reset the value using the above API.

Performance

If you're considering inserting large amounts of data, try to take advantage of inserting multiple Level 1 rows. Of course, the panel group you're accessing must have that particular panel architecture. By “batching” data up in this fashion, you save less trips across the network.

Take a look at indexing. Verify the search keys being used line up with the indexes created for a table. Say that you have a record that contains indexes to fields A, B, C, D and E in that order. Say that you have a search record with values for fields A, C and E. It'll take a long time to look for a record with these key criteria because the database will have to perform a full index table scan. If you have a search record with values for fields A, B and C, it will perform the lookup a lot quicker. Refer to your DBMS manuals.

Message Agent API

C Function API Specifics

To use the C API, you need these files:

- PSMSG.DLL contains the C function API for communicating with the Message Agent (PSMAG.EXE). It must be available on the machine running the Message Agent.
- PSMSG.LIB is an import library you can link to in order to call PSMSG.DLL.
- PSMSGAPI.H contains the function prototypes and definitions for the API.

OLE Automation Specifics

The PeopleSoft installation program registers the Message Agent as an OLE automation server. However, if you want to make sure the registration was done correctly, you can enter this command line from the PeopleTools BIN directory:

```
REGSVR32 PSMAS.DLL
```

The PeopleTools.MessageAgent object has methods corresponding to most of the C API functions. The method names, parameters, and return codes are the same as the C API functions given in the function descriptions, with these exceptions:

- For the method name, use the function name without its PSMsg prefix.
- Leave out the *hContext* parameter.
- For parameters shown as pointers in the C syntax, use reference variables.
- These functions are not available:
 - GetEditTableFieldList
 - GetListBoxField
 - GetListBoxRow
 - GetPromptValueRow
 - GetSearchList
 - SetOptions

Operations by Functional Category

Session Level Operations

CheckAndSetOperator	Sets the Operator ID the Message Agent uses to log into the PeopleSoft database.
Connect	Verifies that the Application Server port is available.
Disconnect	Destroys client side objects.

Processing Messages

StartMessage	Identifies message definition to be used.
--------------	---

ProcessMessage	Triggers message processing after all input message fields have been sent.
----------------	--

Field Level Operations

SetField	Sets the value of a message field to be mapped into a panel buffer when the message is processed.
GetMaxFieldNameLength	Determines the length of the longest field name in the output message field list.
GetMaxValueLength	Determines the length of the longest value in the output message field list.
FindField	Locates a specified field by name in the output message field list and sets the current position to that field.
FindFirstField	Locates the first field in the output message field list and sets the current list position to that field.
FindNextField	Locates the next field in the output message field list and sets the current list position to that field.
FindNextOutputRow	Moves the current position to the next data row in the output list.
GetFieldCount	Returns the number of fields in the message definition.
GetFieldInfo (OLE only)	Returns information about a field in the message definition.
GetFieldList (C only)	Returns a list of the fields in the message definition, along with their information.
GetFieldNameLength	Determines the length of the current field name in the output message field list.
GetFieldName	Retrieves the field name of the current field in the output message field list.
GetValueLength	Determines the length of the value in the current output message field.
GetValue	Retrieves the value for the current entry in the output message field list.

Error Processing

GetErrorExplainText	Gets the detailed description associated with a PeopleSoft error message set number and message number.
GetErrorExplainTextLength	Returns the length of the detailed description associated with a PeopleSoft error message.
GetErrorFieldName	Returns the name of the record field that was the source of an error.
GetErrorFieldNameLength	Returns the length of the field name associated with a PeopleSoft error message.
GetErrorRecordName	Returns the name of the record definition that was the source of an error.
GetErrorRecordNameLength	Returns the length of the record definition name associated with a PeopleSoft error message.
GetErrorText	Retrieves the error text for the first error encountered while processing a message.
GetErrorTextLength	Returns the length of the error text.

Search Dialog Processing

GetSearchFieldCount	Returns the number of fields in the message definition's search record definition.
GetSearchFieldInfo (OLE only)	Returns the attributes for a field in the message definition's search record.
GetSearchList (C only)	Returns the attributes for the fields in the message definition's search record.
GetSearchRecord	Retrieves the search record definition for the current message definition.
GetSearchRecordLength	Determines the length of the record definition name for the current message definition's search record.
ProcessSearchDialog	Searches the PeopleSoft database using the values set in the search record fields.

List Box Processing

FindFirstListBoxField	Locates the first field in the field list returned by a
-----------------------	---

	search and sets the current position there.
FindFirstListBoxRow	Locates the first record returned in a search.
FindListBoxField	Locates a field in a record returned by a search and sets the current position to that field.
FindListBoxRow	Sets the current position in a list box to a specified row.
FindNextListBoxField	Locates the next field in the field list returned by a search and sets the current list position to that field.
FindNextListBoxRow	Locates the next record returned in a search.
GetListBoxField (C only)	Returns information about the current field in the search result list box.
GetListBoxFieldInfo (OLE only)	Returns information about the current field in the search result list box.
GetListBoxFieldCount	Returns the number of fields in the record definition for a search result list box.
GetListBoxRow (C only)	Returns the contents of the current list box row.
GetListBoxRowCount	Returns the number of rows (database records) in a search result list box.

Edit Table Processing

FindFirstPromptValueRow	Locates the first record in a list of prompt values.
FindNextPromptValueRow	Locates the next row in a list of prompt values.
FindPromptValueRow	Sets the current position in a prompt value list to a specified row.
GetEditTableFieldCount	Returns the number of fields in a prompt table.
GetEditTableFieldInfo (OLE only)	Returns field information for a field in a prompt table.
GetEditTableFieldList (C only)	Returns field information for the fields in a prompt table.
GetPromptValueInfo (OLE only)	Returns the contents of the current field from a prompt table.
GetPromptValueFieldCount	Returns the number of fields in a prompt

	value result set.
GetPromptValueRow (C only)	Returns the contents of the current row from a prompt table.
GetPromptValueRowCount	Returns the number of rows retrieved from a prompt table.
ProcessPromptTable	Retrieves values from a field's prompt table.

CheckAndSetOperator

Syntax

```
int stdcall PSMsgCheckAndSetOperator(PMSGHANDLE hContext, LPPSOPERATOR
lpOperator);
```

Description

Sets the Operator ID the Message Agent uses to log into the PeopleSoft database.

CheckAndSetOperator enables you to set the Operator ID that the Message Agent uses to log into the PeopleSoft database. The Operator ID determines the Message Agent's security access, both to the data in the database and the menus in PeopleSoft applications. If the Message Agent is already logged in, CheckAndSetOperator changes to the Operator ID specified in the PSOPERATOR structure.



If you change operators using CheckAndSetOperator, the menu security for the new operator takes effect after you change windows. While you remain in the current window, the available menu options are those for the Operator ID in effect before this function call. However, the data security for the new operator, including row-level security, takes effect immediately.

Note that CheckAndSetOperator works only if the PeopleSoft PSOPRDEFN table includes the specified Operator ID. The Message Agent cannot use the custom user exit feature to check security using the PSGETLOGONINFO function.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpOperator</i>	A pointer to a buffer that holds the PeopleSoft Operator ID and password. The form of the structure containing the data is:

```
typedef struct /* logon information structure */
```

```

{
    char    szOprId[PSMSG_OPERATORLEN + 1]; // Operator ID
    char    szOprPswd[PSMSG_OPERATORLEN + 1]; // Password
} PSOPERATOR;

```

Return Value

Value	Meaning
PSMSG_OK	The operator information was successfully set.
PSMSG_NOTFOUND	The specified Operator ID is invalid.
PSMSG_ERROR	A general error occurred.

Connect

Syntax

```

int stdcall LPPSOPERATOR(long lMsgAPIVersion, LPCSTR lpccszTopic, lpoperator
PSMSGHANDLE FAR * lphContext);

```

Description

Identifies Application Server port availability.

Parameters

<i>lMsgAPIVersion</i>	The API version number defined in PSMSGAPI.H. Pass the symbol PSMSG_APIVERSION for this argument.
<i>lphContext</i>	A pointer to a buffer to receive a context handle from the Message Agent.
<i>lpccszTopic</i>	Application Server name or IP address and port number.
<i>Lpoperator</i>	Operator id and password

Return Value

Value	Meaning
PSMSG_OK	The application has successfully connected to the Message Agent.
PSMSG_ERROR	General error.
PSMSG_NOMOREAGENTS	All Message Agent instances using the specified topic name are busy.

PSMSG_NOTLOADED	Message Agent is not present.
-----------------	-------------------------------

Disconnect

Syntax

```
int stdcall PSMsgDisconnect(PSMSGHANDLE hContext);
```

Description

Disconnects the calling program from the Message Agent.

Parameters

hContext The context handle assigned when this program connected to the Message Agent.

Return Value

Value	Meaning
PSMSG_OK	The application successfully disconnected from the Message Agent.
PSMSG_ERROR	General error.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE.

FindField

Syntax

```
int stdcall PSMsgFindField(PSMSGHANDLE hContext, LPCSTR lpcszFieldName);
```

Description

Locates a field in the output message field list and sets the current position to that field.

After the Message Agent processes a message with `ProcessMessage`, it builds a list of the output message fields defined in the Application Designer message definition. If your application is looking for a specific output field, use `FindField` to set the current list position to that field, then use `GetValue` to extract the field's value.

Parameters

hContext The context handle.

lpcszFieldName The name of the field to find. The field must be designated as an output field in the message definition.

Return Value

Value	Meaning
PSMSG_OK	Success. The location is set to the field.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg.
PSMSG_NOTFOUND	The requested field was not found.

FindFirstField**Syntax**

```
int stdcall PSMsgFindFirstField(PSMSGHANDLE hContext);
```

Description

Locates the first field in the output message field list and sets the current position there.

After the Message Agent processes a message with `ProcessMessage`, it builds a list of the output message fields defined in the Application Designer message definition. `FindFirstField` sets the current list position to the beginning of the list. Use `GetFieldName` and `GetValue` to extract the field name and value, then use `FindNextField` to move to the next field.

If your application needs a specific output field, use `FindField` to find the field by name.

Parameters

hContext The context handle.

Return Value

Value	Meaning
PSMSG_OK	Success. The location is set to the first field.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE.
PSMSG_NOTFOUND	Output field list was empty.

FindFirstListBoxField**Syntax**

```
int stdcall PSMsgFindFirstListBoxField(PSMSGHANDLE hContext);
```


Return Value

<i>Value</i>	<i>Meaning</i>
PSMSG_OK	Success. A row of data is available.
PSMSG_NOTFOUND	The search didn't return any rows.

FindFirstPromptValueRow

Syntax

```
int stdcall PSMsgFindFirstPromptValueRow(PSMSGHANDLE hContext);
```

Description

Locates the first record in a list of prompt values.

When the Message Agent processes the edit table for a field, it returns a list of the rows from the edit table meeting the search criteria.

To retrieve values from this table, you first set the current row using one of two methods: move to a specific row in the result set using FindPromptValueRow, or systematically scroll through the rows using FindFirstPromptValueRow and FindNextPromptValueRow.

With FindPromptValueRow, you specify which row you want by its order in the list. The first row is row 0, the second is row 1, and so on. To determine how many rows are available, use GetPromptValueRowCount.

Once you've set the current row, you can retrieve field values from the row using GetPromptValueRow.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
-----------------	---

Return Value

<i>Value</i>	<i>Meaning</i>
PSMSG_OK	Success. A row of data is available.
PSMSG_NOTFOUND	The search didn't return any rows.

FindListBoxField

Syntax

```
int stdcall PSMsgFindListBoxField(PSMSGHANDLE hContext, LPCSTR lpcszFieldName);
```

Description

Locates a field in a record returned by a search and sets the current position to that field.

When the Message Agent processes the search dialog box, it returns a “table” of values, where each row is a record meeting the search criteria and each column is a field specified as a list box field in the search record definition.

To retrieve values from this table, you first set the current row using one of two methods: move to a specific row in the result set using `FindListBoxRow`, or systematically scroll through the rows using `FindFirstListBoxRow` and `FindNextListBoxRow`.

Once you’ve set the current row, you can retrieve field values from the row using one of three methods:

- Copy the entire row into an array using `GetListBoxRow`
- Systematically move from one field to the next with `FindFirstListBoxField` and `FindNextListBoxField`, using `GetListBoxField` to retrieve the value of each field
- Use `FindListBoxField` to specify which field’s value you want, then use `GetListBoxField` to retrieve the value

Parameters

<i>hContext</i>	The context handle.
<i>lpcszFieldName</i>	The name of the field to find. The specified field must be a field in the search record definition for the current message definition.

Return Value

<i>Value</i>	<i>Meaning</i>
PSMSG_OK	Success. The location is set to the field.
PSMSG_NOTFOUND	The requested field was not found.

FindListBoxRow

Syntax

```
int stdcall PSMsgFindListBoxRow(PMSGHANDLE hContext int nRowIndex);
```

Description

Sets the current position in a list box to a specified row.

When the Message Agent processes the search dialog box, it returns a “table” of values, where each row is a record meeting the search criteria and each column is a field specified as a list box field in the search record definition.

To retrieve values from this table, you first set the current row using one of two methods: move to a specific row in the result set using `FindListBoxRow`, or systematically scroll through the rows using `FindFirstListBoxRow` and `FindNextListBoxRow`.

With `FindListBoxRow`, you specify which row you want by its order in the list box. The first row is row 0, the second is row 1, and so on. To determine how many rows are available, use `GetListBoxRowCount`.

Once you've set the current row, you can retrieve field values from the row using one of three methods:

- Copy the entire row into an array using `GetListBoxRow`
- Systematically move from one field to the next with `FindFirstListBoxField` and `FindNextListBoxField`, using `GetListBoxField` to retrieve the value of each field
- Use `FindListBoxField` to specify which field's value you want, then use `GetListBoxField` to retrieve the value

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>nRowIndex</i>	An integer indicating a row within the list box.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_NOTFOUND	The specified row was not found.

FindNextField

Syntax

```
int stdcall PSMsgFindNextField(PSMSGHANDLE hContext);
```

Description

Locates the next field in the output message field list and sets the current list position to that field.

Use `FindNextField` in conjunction with `FindFirstField` to visit each field in the output message field list. For each field, use `GetFieldName` and `GetValue` to extract the field name and value. If your application only needs particular output fields, use `FindField` to locate fields by their names.

Parameters

hContext The context handle assigned when this program connected to the Message Agent.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE.
PSMSG_NOTFOUND	There was no next field in the output list.

FindNextListBoxField

Syntax

```
int stdcall PSMsgFindNextListBoxField(PSMSGHANDLE hContext);
```

Description

Locates the next field in the field list returned by a search and sets the current list position to that field.

When the Message Agent processes the search dialog box, it returns a “table” of values, where each row is a record meeting the search criteria and each column is a field specified as a list box field in the search record definition.

To retrieve values from this table, you first set the current row using one of two methods: move to a specific row in the result set using FindListBoxRow, or systematically scroll through the rows using FindFirstListBoxRow and FindNextListBoxRow.

Once you’ve set the current row, you can retrieve field values from the row using one of three methods:

- Copy the entire row into an array using GetListBoxRow
- Systematically move from one field to the next with FindFirstListBoxField and FindNextListBoxField, using GetListBoxField to retrieve the value of each field
- Use FindListBoxField to specify which field’s value you want, then use GetListBoxField to retrieve the value

Parameters

hContext The context handle assigned when this program connected to the Message Agent.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_NOTFOUND	There was no next field in the list.

FindNextListBoxRow**Syntax**

```
int stdcall PSMsgFindNextListBoxRow(PMSGHANDLE hContext);
```

Description

Locates the next record returned in a search.

When the Message Agent processes the search dialog box, it returns a “table” of values, where each row is a record meeting the search criteria and each column is a field specified as a list box field in the search record definition.

To retrieve values from this table, you first set the current row using one of two methods: move to a specific row in the result set using FindListBoxRow, or systematically scroll through the rows using FindFirstListBoxRow and FindNextListBoxRow.

Once you’ve set the current row, you can retrieve field values from the row using one of three methods:

- Copy the entire row into an array using GetListBoxRow
- Systematically move from one field to the next with FindFirstListBoxField and FindNextListBoxField, using GetListBoxField to retrieve the value of each field
- Use FindListBoxField to specify which field’s value you want, then use GetListBoxField to retrieve the value

Parameters

hContext The context handle assigned when this program connected to the Message Agent.

Return Value

Value	Meaning
PSMSG_OK	Success. A row of data is available.
PSMSG_NOTFOUND	There are no more rows of data.

FindNextOutputRow

Syntax

```
int stdcall PSMsgFindNextOutputRow(PMSGHANDLE hContext);
```

Description

Locates the next row of output for a message that maps data from a level one scroll.

The Message Agent can map data to and from fields that appear inside a level 1 scroll on a panel. When a scroll includes more than one row of data, the Message Agent creates a “table” of output values. FindNextOutputRow moves the current position to the next row in the table, so that you can retrieve data from the fields in that row.

When you start retrieving data, the initial current row is the first row in the scroll. So the *first* use of FindNextOutputRow advances the current position to the *second* row.

Once you’ve sent this message, the other Find messages (FindField, FindFirstField, and FindNextField) set the current position to fields in the current row. Use GetFieldName and GetValue to extract the field name and value.



This function is only available for level 1 scrolls.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
-----------------	---

Return Value

Value	Meaning
PSMSG_OK	Success. Another row of data is available.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PMSGHANDLE passed to PSMsg.
PSMSG_NOTFOUND	There are no more rows of data in the scroll.

FindNextPromptValueRow

Syntax

```
int stdcall PSMsgFindNextPromptValueRow(PMSGHANDLE hContext);
```

Description

Locates the next row in a list of prompt values.

When the Message Agent processes the edit table for a field, it returns a list of the rows from the edit table meeting the search criteria.

To retrieve values from this table, you first set the current row using one of two methods: move to a specific row in the result set using `FindPromptValueRow`, or systematically scroll through the rows using `FindFirstPromptValueRow` and `FindNextPromptValueRow`.

With `FindPromptValueRow`, you specify which row you want by its order in the list. The first row is row 1, the second is row 2, and so on. To determine how many rows are available, use `GetPromptValueRowCount`.

Once you've set the current row, you can retrieve field values from the row using `GetPromptValueRow`.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
-----------------	---

Return Value

<i>Value</i>	<i>Meaning</i>
PSMSG_OK	Success. A row of data is available.
PSMSG_NOTFOUND	There are no more rows of data.

FindPromptValueRow

Syntax

```
int stdcall PSMsgFindPromptValueRow(PSMSGHANDLE hContext int nRowIndex);
```

Description

Sets the current position in a prompt value list to a specified row.

When the Message Agent processes the edit table for a field, it returns a list of the rows from the edit table meeting the search criteria.

To retrieve values from this table, you first set the current row using one of two methods: move to a specific row in the result set using `FindPromptValueRow`, or systematically scroll through the rows using `FindFirstPromptValueRow` and `FindNextPromptValueRow`.

With `FindPromptValueRow`, you specify which row you want by its order in the list. The first row is row 1, the second is row 2, and so on. To determine how many rows are available, use `GetPromptValueRowCount`.

Once you've set the current row, you can retrieve field values from the row using `GetPromptValueRow`. There are no functions for navigating through the fields in the row.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>nRowIndex</i>	An integer indicating a row within the list of prompt values.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_NOTFOUND	The row could not be found.

GetEditTableFieldCount

Syntax

```
int stdcall PSMsgGetEditTableFieldCount(PMSMSGHANDLE hContext, LPCSTR  
lppszRecordName, LPINT lpnCount);
```

Description

Returns the number of fields in a prompt table.

`GetEditTableFieldCount` returns the number of fields in an edit table. You can use this information to allocate a buffer large enough for `GetEditTableFieldList` to copy the field attributes into.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lppszRecordName</i>	A pointer to a buffer containing the name of the record definition for the edit table. Do not include the PS_prefix from the table name.
<i>lpnCount</i>	A pointer to a buffer to receive the number of fields.

Return Value

PSMSG_OK

GetEditTableFieldInfo

OLE Method

```
long GetEditTableFieldInfo(LPCTSTR lpszRecordName, long nFieldNum, LPDISPATCH  
FAR *lpSearchObj);
```

Description

Returns field information for a field in a prompt table. (OLE Automation only; for C, use GetEditTableFieldList)

When you define a field in the Application Designer, you have the option to assign it an *edit table*, also known as a *prompt table*. The edit table lists valid values for the field; when you enter data, the system checks it against the edit table. In the online system, the user can press F4 to display the contents of the edit table and select a value.

If a field has an edit table associated with it, the edit table name appears as the last piece of data in the field attribute object for that field (szEditTable). Note that the name is the name of the record definition for the table, not the name of the table itself; it doesn't include the PS_ prefix.

GetEditTableFieldInfo creates an object with all the information you need about a field in the edit table.

Parameters

<i>lpszRecordName</i>	A pointer to a buffer containing the name of the record definition for the edit table. Do not include the PS_ prefix from the table name.
<i>nFieldNum</i>	The number of the field in the edit table field you want.
<i>lpSearchObj</i>	A pointer to an object in which the function will specify field information. The properties of the object are:

```

SzFieldName (string)
SzLongName (string)
SzShortName (string)
FFieldType (integer) // Field type. Valid values are
                        //
PSMSG_TYPE_CHAR, PSMSG_TYPE_NUMBER,
                        //
PSMSG_TYPE_DATE, PSMSG_TYPE_TIME,
                        //
PSMSG_TYPE_DATETIME,
// PSMSG_TYPE_UNSUPPORTED
nFieldSize (integer)
fFieldUse (integer) // Field use attribute(s). Values are
                        //
PSMSG_USE_KEY, PSMSG_USE_DUPLKEY,
                        //
PSMSG_USE_ALTKEY, PSMSG_USE_DESCKEY,
                        //PSMSG_USE_SEARCHITEM,PSMSG_USE_L
ISTITEM
szEditTable (string) // Edit table

```

Return Value

<i>Value</i>	<i>Meaning</i>
PSMSG_OK	Success.
PSMSG_ERROR	The specified edit table does not exist.

GetEditTableFieldList

Syntax

```
int stdcall PSMsgGetEditTableFieldList(PSMSGHANDLE hContext, LPCSTR
lpcszRecordName, PSMSGSEARCHFIELD lpFieldInfo);
```

Description

Returns field information for the fields in a prompt table.



This function is not available through the OLE Automation interface. Use `GetEditTableFieldInfo` instead.

When you define a field in the Application Designer, you have the option to assign it an *edit table*, also known as a *prompt table*. The edit table lists valid values for the field; when you enter data, the system checks it against the edit table. In the online system, the user can press F4 to display the contents of the edit table and select a value.

If a field has an edit table associated with it, the edit table name appears as the last piece of data in the field attribute structure for that field (`szEditTable`). Note that the name is the name of the record definition for the table, not the name of the table itself; it doesn't include the `PS_` prefix.

`GetEditTableFieldList` creates an array with all the information you need about the fields in the edit table. Each element of the array is a structure as shown above; the number of elements in the array is the number of fields in the edit table, which you can determine using `GetEditTableFieldCount`.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpszRecordName</i>	A pointer to a buffer containing the name of the record definition for the edit table. Do not include the <code>PS_</code> prefix from the table name.
<i>lpFieldInfo</i>	A pointer to a buffer in which the function will create an array of field attribute structures. The form of the structure is:

typedefstruct

```
{
    char  szFieldName[PSMSG_FIELDNAMELEN + 1]; // Field name
    char  szLongName[PSMSG_LONGNAMELEN + 1];  // Long name
    char  szShortName[PSMSG_SHORTNAMELEN + 1]; // Short name
    INT   fFieldType;                          // Field type. Valid values are
                                                // PSMSG_TYPE_CHAR, PSMSG_TYPE_NUMBER,
                                                // PSMSG_TYPE_DATE, PSMSG_TYPE_TIME,
                                                // PSMSG_TYPE_DATETIME,
                                                // PSMSG_TYPE_UNSUPPORTED
    INT   nFieldSize;                          // Field size in bytes
}
```

```

INT    fFieldUse;           // Field use attribute(s). Values are
                             // PMSGMSG_USE_KEY, PMSGMSG_USE_DUPLKEY,
                             // PMSGMSG_USE_ALTKEY, PMSGMSG_USE_DESCKEY,
                             //PMSGMSG_USE_SEARCHITEM,PMSGMSG_USE_LISTITEM
char    szEditTable[PMSGMSG_RECNAMELEN + 1]: // Edit table
} PMSGMSGSEARCHFIELD;

```

Return Value

<i>Value</i>	<i>Meaning</i>
PMSGMSG_OK	Success.
PMSGMSG_ERROR	The specified edit table does not exist.

GetErrorExplainText

Syntax

```

int stdcall PSMsgGetErrorExplainText(PMSGMSGHANDLE hContext, LPCSTR
lppszExplainText, int nSize);

```

Description

Gets the detailed description associated with a PeopleSoft error message.

When an PeopleSoft application user receives an error message, the message box includes an **Explain** button. The user clicks the button to view more a more detailed description of the problem. GetErrorExplainText enables the Message Agent to retrieve the same detailed description. It retrieves the descriptive text associated with the first error the Message Agent encountered while processing a message.

Use GetErrorExplainText in conjunction with GetErrorText. If the GetErrorText message includes a message set and message number at the end, GetErrorExplainText returns the “explain” text for that set and message.

To determine how large the descriptive text is, and consequently how large a buffer to allocate, use GetErrorExplainTextLength.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lppszExplainText</i>	A pointer to a buffer to receive the error text.

nSize The maximum number of characters to copy to the buffer, including the null terminator. The error text is truncated if it is longer than *nSize*.

Return Value

PSMSG_OK, even if the message doesn't include any "explain" text.

GetErrorExplainTextLength

Syntax

```
int stdcall PSMsgGetErrorExplainTextLength(PMSGHANDLE hContext, LPINT  
lpnLength);
```

Description

Returns the length of the detailed description associated with a PeopleSoft error message.

GetErrorExplainTextLength determines the length of the descriptive text associated with the first error the Message Agent encountered while processing a message. This length does not include the null terminator for the string.

You can use this information to allocate a buffer large enough to hold the text. Use GetErrorExplainText to retrieve the actual text.

Parameters

hContext The context handle assigned when this program connected to the Message Agent.

lpnLength A pointer to a buffer to receive the text length.

Return Value

PSMSG_OK, even if the message doesn't include any descriptive text.

GetErrorFieldName

Syntax

```
int stdcall PSMsgGetErrorFieldName(PMSGHANDLE hContext, LPCSTR  
lppszFieldName, int nSize);
```

Description

Returns the name of the record field that was the source of an error.

When the data the Message Agent enters into a panel fails an online edit, you can use `GetErrorFieldName` to determine which record field the Message Agent wasn't able to update. If more than one field failed an edit, `GetErrorFieldName` returns the name of the first field to fail.

To determine how large the field name is, use `GetErrorFieldNameLength`. To determine which record definition contains the field, use `GetErrorRecordName`.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpsczFieldName</i>	A pointer to a buffer to receive the field name.
<i>nSize</i>	The maximum number of characters to copy to the buffer, including the null terminator. The record definition name is truncated if it is longer than <i>nSize</i> .

Return Value

Value	Meaning
PSMSG_OK	The field name was successfully set.
PSMSG_ERROR	The field name could not be determined.

GetErrorFieldNameLength

Syntax

```
int stdcall PSMsgGetErrorFieldNameLength(PSMSGHANDLE hContext, LPINT
lpnLength);
```

Description

Returns the length of the field name associated with a PeopleSoft error message.

`GetErrorFieldNameLength` determines the length of the field name associated with the first error the Message Agent encountered while processing a message. This length does not include the null terminator for the string.

You can use this information to allocate a buffer large enough to hold the text. Use `GetErrorFieldName` to retrieve the actual field name.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnLength</i>	A pointer to a buffer to receive the length of the field name.

Return Value

Value	Meaning
PSMSG_OK	The field name length was successfully set.
PSMSG_ERROR	The field for the message could not be determined.

GetErrorRecordName

Syntax

```
int stdcall PSMsgGetErrorRecordName(PSMSGHANDLE hContext, LPCSTR  
lppszRecordName, int nSize);
```

Description

Returns the name of the record definition that was the source of an error.

When the data the Message Agent enters into a panel fails an online edit, you can use `GetErrorRecordName` to determine which record definition contains the field the Message Agent wasn't able to update. If more than one field failed an edit, `GetErrorRecordName` returns the record definition name for the first failure encountered.

To determine how large the record definition name is, use `GetErrorRecordNameLength`. To determine which record field failed the edits, use `GetErrorFieldName`.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lppszRecordName</i>	A pointer to a buffer to receive the record definition name.
<i>nSize</i>	The maximum number of characters to copy to the buffer, including the null terminator. The record definition name is truncated if it is longer than <i>nSize</i> .

Return Value

Value	Meaning
PSMSG_OK	The record definition name was successfully set.
PSMSG_ERROR	The record definition name could not be determined.

GetErrorRecordNameLength

Syntax

```
int stdcall PSMsgGetErrorRecordNameLength(PMSGHANDLE hContext, LPINT  
lpnLength);
```

Description

Returns the length of the record definition name associated with a PeopleSoft error message.

GetErrorRecordNameLength determines the length of the record definition name associated with the first error the Message Agent encountered while processing a message. This length does not include the null terminator for the string.

You can use this information to allocate a buffer large enough to hold the text. Use GetErrorRecordName to retrieve the actual text.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnLength</i>	A pointer to a buffer to receive the length of the record definition name.

Return Value

Value	Meaning
PSMSG_OK	The record definition name length was successfully set.
PSMSG_ERROR	The record definition for the error message could not be determined.

GetErrorText

Syntax

```
int stdcall PSMsgGetErrorText(PMSGHANDLE hContext,  
LPSTR lpzErrorText,int nSize);
```

Description

Retrieves the error text for the first error encountered while processing a message.

When the Message Agent encounters an error, it records a textual description of the error and goes into an error state. Once in the error state, the Message Agent will only respond to StartMessage, GetErrorText, GetErrorTextLength, and Disconnect.

To determine how large the error text buffer needs to be, use GetErrorTextLength.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpSzErrorText</i>	A pointer to a buffer to receive the error text.
<i>nSize</i>	The maximum number of characters to copy to the buffer, including the null terminator. The error text is truncated if it is longer than the number of characters in <i>nSize</i> .

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE.

GetErrorTextLength

Syntax

```
int stdcall PSMsgGetErrorTextLength(PSMSGHANDLE hContext,  
LPINT lpnLength);
```

Description

Returns the length of the error text.

GetErrorTextLength determines the length of the error string for the first error the Message Agent encountered while processing a message. This length does not include the null terminator for the string.

You can use this information to allocate a buffer large enough to hold the error text. Use GetErrorText to retrieve the actual message.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnLength</i>	A pointer to a buffer to receive the error text length.

Return Value

Value	Meaning
PSMSG_OK	Success.

PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg function.

GetFieldCount

Syntax

```
int stdcall PSMsgGetFieldCount(PSMSGHANDLE hContext, LPINT lpnCount);
```

Description

Returns the number of fields mapped in the message definition.

You can use the Message Agent to get field information and create an interface for the user to enter data. GetFieldCount writes the number of fields in the message definition mapping at the location specified by the *lpnCount* buffer; GetFieldList returns the field information.

Parameters

hContext The context handle assigned when this program connected to the Message Agent.

lpnCount A pointer to a buffer to receive the number of fields.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg function.

GetFieldInfo

OLE Method Syntax

```
long GetFieldInfo(long nFieldNum, LPDISPATCH FAR *FieldObject);
```

Description

Returns a list of the fields in the message definition. (OLE Automation method only; for C, use GetFieldList instead)

You can use the Message Agent to get field information and create an interface for the user to enter data. GetFieldInfo provides the information you need about a field in the mapping to create an appropriate control.

Parameters

nFieldNum The number of the field in the message definition whose information you want.

FieldObject A pointer to the IDispatch interface of a field object. The object has these properties:

```
szFieldName(string)    // Field name

fFieldType(integer)    // Field type. Valid values are
                        // PSMSG_TYPE_CHAR, PSMSG_TYPE_NUMBER,
                        // PSMSG_TYPE_DATE, PSMSG_TYPE_TIME,
                        // PSMSG_TYPE_DATETIME,
                        // PSMSG_TYPE_UNSUPPORTED

nFieldSize(integer)    // Field size in bytes

fFieldMapUse(integer)  // Field use attribute(s). Values are
                        // PSMSG_USE_INPUT, PSMSG_USE_OUTPUT,
                        // PSMSG_USE_BOTH, PSMSG_USE_SEARCHKEY

fFieldMapTransfer(integer) // Field method attribute(s).
                        // Values are
                        // PSMSG_XFR_COPY, PSMSG_XFR_XLATS,
                        // PSMSG_XFR_XLATL, PSMSG_XFR_REVERSE

nFieldLevel(integer)  // Field's scroll level
```

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	Error.

GetFieldList

Syntax

```
int stdcall PSMsgGetFieldList(PMSGHANDLE hContext,
                              LPPMSGFIELDINFO lpFields);
```

Description

Returns a list of the fields in the message definition.



This function is not available through the OLE Automation interface. Use GetFieldInfo instead.

You can use the Message Agent to get field information and create an interface for the user to enter data. GetFieldList provides the information you need about each field in the mapping to create an appropriate control. GetFieldCount returns the number of fields in the mapping.

Parameters

hContext The context handle assigned when this program connected to the Message Agent.

lpFieldst A pointer to a buffer to receive the field information. The form of the structure is:

typedef struct

```
{
    char  szFieldName[PSMSG_FIELDNAMELEN + 1]; // Field name
    INT   fFieldType;           // Field type. Valid values are
                                // PSMSG_TYPE_CHAR, PSMSG_TYPE_NUMBER,
                                // PSMSG_TYPE_DATE, PSMSG_TYPE_TIME,
                                // PSMSG_TYPE_DATETIME,
                                // PSMSG_TYPE_UNSUPPORTED
    INT   nFieldSize;           // Field size in bytes
    INT   fFieldMapUse;         // Field use attribute(s). Values are
                                // PSMSG_USE_INPUT, PSMSG_USE_OUTPUT,
                                // PSMSG_USE_BOTH, PSMSG_USE_SEARCHKEY
```

```

INT    fFieldMapTransfer;    // Field method attribute(s). Values are
                                // PSMSG_XFR_COPY, PSMSG_XFR_XLATS,
                                // PSMSG_XFR_XLATL, PSMSG_XFR_REVERSE

INT    nFieldLevel;          // Field's scroll level

} PSMSGFIELDINFO;

```

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg function.

GetFieldName

Syntax

```

int stdcall PSMsgGetFieldName(PSMSGHANDLE hContext,
                              LPSTR lpszFieldName, int nSize);

```

Description

Retrieves the field name of the current field in the output message field list.

After the Message Agent processes a message with ProcessMessage, it builds a list of the output message fields defined in the Application Designer message definition. GetFieldName returns the name of the current field in that list.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpszValue</i>	A pointer to a buffer to receive the field name of the output message field.
<i>nSize</i>	The maximum number of characters to copy to the buffer, including the null terminator. The field name is truncated if it is longer than the number of characters in <i>nSize</i> .

Return Value

Value	Meaning
PSMSG_OK	Success.

PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg function.

GetFieldNameLength

Syntax

```
int stdcall PSMsgGetFieldNameLength(PSMSGHANDLE hContext, LPINT lpnLength);
```

Description

Determines the length of the current field name in the output message field list.

After the Message Agent processes a message with ProcessMessage, it builds a list of the output message fields defined in the Application Designer message definition. GetFieldNameLength returns the length of the name of the current field in that list. This length does not include the null terminator for the string.

You can use this information to allocate a buffer large enough to hold the current field name. To determine the length of the longest output field name, use GetMaxFieldNameLength.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnLength</i>	A pointer to a buffer for the field name length.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg function.

GetListBoxField

Syntax

```
int stdcall PSMsgGetListBoxField(PSMSGHANDLE hContext, LPLISTBOXFIELDINFO lpFieldInfo);
```

Description

Returns information about the current field in the search result list box.



This function is not available through the OLE Automation interface. Use GetListBoxFieldInfo instead.

When the Message Agent processes the search dialog box, it returns a “table” of values, where each row is a record meeting the search criteria and each column is a field specified as a list box field in the search record definition.

To retrieve values from this table, you first set the current row using one of two methods: move to a specific row in the result set using FindListBoxRow, or systematically scroll through the rows using FindFirstListBoxRow and FindNextListBoxRow.

Once you’ve set the current row, you can retrieve field values from the row using one of three methods:

- Copy the entire row into an array using GetListBoxRow
- Systematically move from one field to the next with FindFirstListBoxField and FindNextListBoxField, using GetListBoxField to retrieve the value of each field
- Use FindListBoxField to specify which field’s value you want, then use GetListBoxField to retrieve the value

Parameters

hContext The context handle assigned when this program connected to the Message Agent.

lpFieldInfo A pointer to a buffer to receive the field information. The form of the structure holding the field information is:

typedef struct

```
{
    char  szFieldName[PSMSG_FIELDNAMELEN + 1]; // Field name
    char  szValue[PSMSG_MAXVALUELEN + 1];      // Field value
    int   nFieldNameSize;                       // Field name size in bytes
    int   nValueSize;                           // Value's size in bytes
} PSMSGLISTBOXFIELDINFO;
```

Return Value

Value	Meaning
PSMSG_OK	The field exists and its attributes have been

	copied into the structure.
PSMSG_NOTFOUND	The specified field does not exist.

GetListBoxFieldCount

Syntax

```
int stdcall PSMsgGetListBoxFieldCount(PSMSGHANDLE hContext, LPINT lpnCount);
```

Description

Returns the number of fields in the record definition for a search result list box.

After the Message Agent processes a search dialog, it returns a list of the database records matching the search criteria. `GetListBoxFieldCount` returns the number of fields in each record or row. Use this number to create an array of `PSMSGLISTBOXFIELDINFO` structures for `GetListBoxRow`.

To get the values from a particular record, use `GetListBoxRow`. To scroll through the records one at a time, use `FindFirstListBoxRow` and `FindNextListBoxRow`.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnCount</i>	A pointer to a buffer to receive the number of fields.

Return Value

PSMSG_OK, even when no fields exist.

GetListBoxFieldInfo

OLE Method

```
long GetListBoxFieldInfo(long nFieldNum, LPDISPATCH FAR *lpRowObj);
```

Description

Returns information about the current field in the search result list box. (OLE Automation only; for C, use `GetListBoxField` or `GetListBoxRow`)

When the Message Agent processes the search dialog box, it returns a “table” of values, where each row is a record meeting the search criteria and each column is a field specified as a list box field in the search record definition.

To retrieve values from this table, you first set the current row using one of two methods: move to a specific row in the result set using `FindListBoxRow`, or systematically scroll through the rows using `FindFirstListBoxRow` and `FindNextListBoxRow`.

Once you've set the current row, use `GetListBoxFieldInfo` to get information about a field in the row.

Parameters

<i>nFieldNum</i>	The number of the field you want from the list box.
<i>lpRowObj</i>	A pointer to an object holding the field information. The properties of the object are: <ul style="list-style-type: none"> SzFieldName (string) szValue (string) nFieldNameSize (integer) nValueSize (integer)

Return Value

Value	Meaning
PSMSG_OK	The field exists and its attributes have been copied into the structure.
PSMSG_NOTFOUND	The specified field does not exist.

GetListBoxRow

Syntax

```
int stdcall PSMsgGetListBoxRow(PSMSGHANDLE hContext,
                               LPPSMSGLISTBOXFIELDINFO lpRowInfo);
```

Description

Returns the contents of the current list box row.



This function is not available through the OLE Automation interface. Use `GetListBoxFieldInfo` instead.

When the Message Agent processes a search record using `ProcessSearchDialog`, it can return multiple records that meet the search criteria. When it does, the Message Agent creates a “table”

of values. The columns in this table are the fields identified as List Box fields when the record was created in the Application Designer. In the online system, the user sees this table as a list of records in a list box.

You use `FindListBoxRow`, `FindFirstListBoxRow`, and `FindNextListBoxRow` to set a current row within this table, then use `GetListBoxRow` to copy the contents of this row into an array.

Each element in the array is a `PSMSGLISTBOXFIELDINFO` structure; the number of elements is the number of fields in specified in the search record definition as list box fields. To determine the number of fields, use `GetListBoxFieldCount`.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpRowInfo</i>	A pointer to an array of field information structures. The form of the structure is:

typedef struct

```
{
    char  szFieldName[PSMSG_FIELDNAMELEN + 1]; // Field name

    char  szValue[PSMSG_MAXVALUELEN + 1];      // Field value
    int   nFieldNameSize;                      // Field name size in bytes
    int   nValueSize;                          // Value's size in bytes

} PSMSGLISTBOXFIELDINFO;
```

Return Value

`PSMSG_OK`

GetListBoxRowCount

Syntax

```
int stdcall PSMsgGetListBoxRowCount(PSMSGHANDLE hContext, LPINT lpnCount);
```

Description

Returns the number of rows (database records) in a search result list box.

After the Message Agent processes a search dialog, it returns a list of the database records matching the search criteria. `GetListBoxRowCount` returns the number of records. You can use this information to set up a processing loop that handles each record in turn.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnCount</i>	A pointer to a buffer to receive the number of rows.

Return Value

PSMSG_OK

GetMaxFieldNameLength**Syntax**

```
int stdcall PSMsgGetMaxFieldNameLength(PMSGHANDLE hContext,
                                       LPINT lpnLength);
```

Description

Determines the length of the longest field name in the output message field list.

After the Message Agent processes a message with ProcessMessage, it builds a list of the output message fields defined in the Application Designer message definition.

GetMaxFieldNameLength returns the length of the longest field name in that list. The length does not include the null terminator. You can use this information to allocate a buffer large enough for the longest field name.

To determine the length of an individual field, use GetFieldNameLength.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnLength</i>	A pointer to a buffer for the field name length.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PMSGHANDLE passed to PSMsg function.

GetMaxValueLength

Syntax

```
int stdcall PSMsgGetMaxValueLength(PMSGHANDLE hContext,
                                   LPINT lpnLength);
```

Description

Determines the length of the longest value in the output message field list.

After the Message Agent processes a message with `ProcessMessage`, it builds a list of the output message fields defined in the Application Designer message definition. `GetMaxValueLength` returns the length of the longest value in that list. The length does not include the null terminator. You can use this information to allocate a buffer large enough for the longest value.

To determine the length of a particular value, use `GetValueLength`.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnLength</i>	A pointer to a buffer for the length.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PMSGHANDLE passed to PSMsg function.

GetPromptValueFieldCount

Syntax

```
int stdcall PSMsgGetPromptValueFieldCount(PMSGHANDLE hContext, LPINT
                                           lpnCount);
```

Description

Returns the number of fields in a prompt value result set.

After the Message Agent processes the edit table for a field using `ProcessPromptTable`, it returns a list of the rows matching the search criteria. `GetPromptValueFieldCount` returns the number of fields in each row.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnCount</i>	A pointer to a buffer to receive the number of fields.

Return Value

PSMSG_OK

GetPromptValueInfo

OLE Method

```
long GetPromptValueInfo(long nFieldNum, LPDISPATCH FAR *lpRowObj);
```

Description

Returns the contents of the current field from an edit table. (OLE Automation only; for C, use `GetPromptValueRow` instead)

When the Message Agent processes an edit table using `ProcessPromptTable`, it can return multiple rows that meet the search criteria. You use `FindPromptValueRow`, `FindFirstPromptValueRow`, and `FindNextPromptValueRow` to set a current row, then use `GetPromptValueInfo` to copy the contents of a field from this row into an array.

Parameters

<i>nFieldNum</i>	The number of the prompt table field you want.
<i>lpRowObj</i>	A pointer to an object holding field information. The properties of the object are: SzFieldName (string) szValue (string) nFieldNameSize (integer) nValueSize (integer)

Return Value

PSMSG_OK

GetPromptValueRow

Syntax

```
int stdcall PSMsgGetPromptValueRow(PSMSGHANDLE hContext,
    LPPMSGGLISTBOXFIELDINFO lpRowInfo);
```

Description

Returns the contents of the current row from an edit table.



This function is not available through the OLE Automation interface. Use GetPromptValueInfo instead.

When the Message Agent processes an edit table using ProcessPromptTable, it can return multiple rows that meet the search criteria. You use FindPromptValueRow, FindFirstPromptValueRow, and FindNextPromptValueRow to set a current row, then use GetPromptValueRow to copy the contents of this row into an array.

Each element in the array is a PSMSGGLISTBOXFIELDINFO structure; the number of elements is the number of fields in the edit table. To determine the number of fields, use GetPromptValueFieldCount.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpRowInfo</i>	A pointer to an array of field information structures. The form of the structure is:

typedef struct

```
{
    char  szFieldName[PSMSG_FIELDNAMELEN + 1]; // Field name
    char  szValue[PSMSG_MAXVALUELEN + 1];      // Field value
    int   nFieldNameSize;                      // Field name size in bytes
    int   nValueSize;                          // Value's size in bytes
} PSMSGGLISTBOXFIELDINFO;
```

Return Value

PSMSG_OK

GetPromptValueRowCount

Syntax

```
int stdcall PSMsgGetPromptValueRowCount(PMSGHANDLE hContext, LPINT  
lpnCount);
```

Description

Returns the number of rows retrieved from a prompt table.

After the Message Agent processes the edit table for a field, it returns a list of the records from the edit table matching the search criteria. GetPromptValueRowCount returns the number of records. You can use this information to set up a processing loop that handles each record in turn.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnCount</i>	A pointer to a buffer to receive the number of rows.

Return Value

PSMSG_OK

GetSearchFieldCount

Syntax

```
int stdcall PSMsgGetSearchFieldCount(PMSGHANDLE hContext, LPINT lpnCount);
```

Description

Returns the number of fields in the message definition's search record definition.

GetSearchFieldCount writes the number of fields in the search record definition at the location specified by the *lpnCount* buffer. You can use this information to allocate an array of PSMSGSEARCHFIELD large enough to hold the field attributes returned by GetSearchList, as in this example:

```
nRet = PSMsgGetSearchFieldCount(hContext, &nFieldCount);  
  
lpSearchInfo = new PSMSGSEARCHFIELD[nFieldCount];
```

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
-----------------	---

lpnCount A pointer to a buffer to receive the number of fields.

Return Value

PSMSG_OK

GetSearchFieldInfo

OLE Method

```
long GetSearchFieldInfo(long nFieldNum, LPDISPATCH FAR *lpSearchObj);
```

Description

Returns the attributes for the fields in the message definition's search record. (OLE Automation only; for C, use GetSearchList instead)

The search record definition determines which fields you can use to search for database records. GetSearchFieldInfo copies all the information you need about a field in the search record into an object. The object has the properties shown below.

Using the information in this object, you can create a user interface control for the field. Use SetField to enter search criteria into one or more of these fields, then use ProcessSearchDialog to perform the search.

Parameters

nFieldNum The number of the field in the search record you want information for.

lpSearchObj A pointer to an object containing the field attributes. The properties of the object are:

szFieldName (string)

szLongName (string)

szShortName (string)

fFieldType (integer) // Field type. Valid values are

// PSMSG_TYPE_CHAR, PSMSG_TYPE_NUMBER,

// PSMSG_TYPE_DATE,

PSMSG_TYPE_TIME,

// PSMSG_TYPE_DATETIME,

// PSMSG_TYPE_UNSUPPORTED

nFieldSize (integer)

```

fFieldUse (integer)    // Field use attribute(s). Values are

                        // PSMSG_USE_KEY, PSMSG_USE_DUPLKEY,

                        // PSMSG_USE_ALTKEY, PSMSG_USE_DESCKEY,

                        // PSMSG_USE_SEARCHITEM,
                        PSMSG_USE_LISTITEM

szEditTable (string) // Edit table

```

Return Value

<i>Value</i>	<i>Meaning</i>
PSMSG_OK	Success.
PSMSG_ERROR	The message definition doesn't have a search record.

GetSearchList

Syntax

```

int stdcall PSMsgGetSearchList(PSMSGHANDLE hContext, LPPMSGSEARCHFIELD
    lpSearchInfo);

```

Description

Returns the attributes for the fields in the message definition's search record.



This function is not available through the OLE Automation interface. Use GetSearchFieldInfo instead.

The search record definition determines which fields you can use to search for database records. GetSearchList copies all the information you need about the fields in the search record into an array. Each element of the array is a structure as shown below; the number of elements in the array is the number of fields in the search record definition, which you can determine using GetSearchFieldCount.

Using the information in this array, you can create a user interface that enables users to enter search criteria. Use SetField to enter search criteria into one or more of these fields, then use ProcessSearchDialog to perform the search.

Parameters

hContext The context handle assigned when this program connected to the Message Agent.

lpSearchInfo

A pointer to an array of field structures containing the attributes of each field in the search record definition. The form of the structure is:

```
typedef struct
{
    char  szFieldName[PSMSG_FIELDNAMELEN + 1]; // Field name

    char  szLongName[PSMSG_LONGNAMELEN + 1]; // Long name
    char  szShortName[PSMSG_SHORTNAMELEN + 1]; // Short name

    INT    fFieldType;           // Field type. Valid values are
                                   // PSMSG_TYPE_CHAR, PSMSG_TYPE_NUMBER,
                                   // PSMSG_TYPE_DATE, PSMSG_TYPE_TIME,
                                   // PSMSG_TYPE_DATETIME,
                                   // PSMSG_TYPE_UNSUPPORTED
    INT    nFieldSize;           // Field size in bytes

    INT    fFieldUse;            // Field use attribute(s). Values are
                                   // PSMSG_USE_KEY, PSMSG_USE_DUPLKEY,
                                   // PSMSG_USE_ALTKEY, PSMSG_USE_DESCKEY,
                                   // PSMSG_USE_SEARCHITEM, PSMSG_USE_LISTITEM

    char  szEditTable[PSMSG_RECNAMELEN + 1]: // Edit table
} PSMSGSEARCHFIELD;
```

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	The message definition doesn't have a search record.

GetSearchRecord

Syntax

```
int stdcall PSMsgGetSearchRecord(PSMSGHANDLE hContext, LPSTR lpszRecordName,
int nSize);
```

Description

Retrieves the search record definition for the current message definition.

Each message definition specifies a record definition to use as the *search record* for the panel group it opens. The search record determines which fields the Message Agent can use to locate database records for opening in the panel group, and which fields it must provide values for to create a new database record. GetSearchRecord returns the name of the search record definition for the current message definition.



You can use this function only after you've set the current message definition with StartMessage.

To determine how large the record definition name is (so you know how large the buffer needs to be), use GetSearchRecordLength.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpszValue</i>	A pointer to a buffer to receive the name of the search record definition.
<i>nSize</i>	The maximum number of characters to copy to the buffer, including the null terminator. The record definition name is truncated if it is longer than the number of characters in <i>nSize</i> .

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_NOTFOUND	No current message definition. Use StartMessage to set the message definition.

GetSearchRecordLength

Syntax

```
int stdcall PSMsgGetSearchRecordLength(PMSGHANDLE hContext, LPINT lpnLength);
```

Description

Determines the length of the record definition name for the current message definition's search record.

GetSearchRecordLength determines the length of the name of the search record definition for the current message definition. This length does not include the null terminator for the string.

You can use this information to allocate a buffer large enough to hold the text. Use GetSearchRecord to retrieve the actual record definition name.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnLength</i>	A pointer to a buffer for the record definition name length.

Return Value

PSMSG_OK, even if the system can't locate the search record definition.

GetValue

Syntax

```
int stdcall PSMsgGetValue(PSMSGHANDLE hContext,
                          LPSTR lpszValue, int nSize);
```

Description

Retrieves the value for the current entry in the output message field list.

After the Message Agent processes a message with ProcessMessage, it builds a list of the output message fields defined in the Application Designer message definition. GetValue returns the value of the current field in that list.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpszValue</i>	A pointer to a buffer to receive the value of the output message field.
<i>nSize</i>	The maximum number of characters to copy to the buffer, including the null terminator. The value is truncated if it is longer than the number of characters in <i>nSize</i> .

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.

PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg function.

GetValueLength

Syntax

```
int stdcall PSMsgGetValueLength(PSMSGHANDLE hContext,
                                LPINT lpnLength);
```

Description

Determines the length of the value in the current output message field.

After the Message Agent processes a message with ProcessMessage, it builds a list of the output message fields defined in the Application Designer message definition. GetValueLength returns the length of the current value in that list. The length does not include the null terminator. You can use this information to allocate a buffer large enough to hold the value.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnLength</i>	A pointer to a buffer for the length.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg function.

ProcessMessage

Syntax

```
int stdcall PSMsgProcessMessage(PSMSGHANDLE hContext,
                                LPINT lpnReplyOption);
```

Description

Triggers message processing after all input message fields have been set.

After you have set all input message fields with SetField, ProcessMessage opens the panel, sets the values for the input fields, saves the panel data, and prepares a list of output values (if any).

The message definition specifies a reply option of None, Reply, or Forward. The calling program can respond to this option in any way that it appropriate. The expected response is: Reply means to send a confirmation to the original sender; Forward means to forward the field values that were actually set back to the sender.

If an error occurs, the Message Agent records the text of the error and enters an error state; see GetLastErrorText for details.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpnReplyOption</i>	A pointer to a buffer for the message reply option. The message reply option indicates how PeopleSoft expects the calling program to respond to the originator of the message.

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMMsg function.
PSMSG_NOMATCHINGROWS	The Message Agent attempted to locate a record in Update/Display mode, but could not locate it.

Reply Option Value

Value	Meaning
PSMSG_NOREPLY	No action.
PSMSG_REPLY	Send confirmation to the originator of the message.
PSMSG_FORWARD	Return a copy of the new field values to the originator.

ProcessPromptTable

Syntax

```
int stdcall PSMMsgProcessPromptTable(PSMSGHANDLE hContext, LPCSTR
lpnRecordName, int nMaxRows);
```

Description

Retrieves values from a field's edit table.

ProcessPromptTable retrieves values from the specified edit table. If you've set values in any of the edit table fields using SetField, the Message Agent uses the values as search criteria, returning only rows with matching values in those fields.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpszRecordName</i>	A pointer to a buffer containing the name of the record definition for the edit table. Do not include the PS_prefix from the table name.
<i>nMaxRows</i>	The maximum number of rows to return. If you specify <i>nMaxRows</i> as 0, the Message Agent will use the row limit for your system (300 by default).

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_NOMATCHINGROWS	No database rows matched the search criteria.
PSMSG_NOTFOUND	The search criteria are not valid.
PSMSG_ERROR	General error.

ProcessSearchDialog

Syntax

```
int stdcall PSMsgProcessSearchDialog(PSMSGHANDLE hContext, int nMaxRows);
```

Description

Searches the PeopleSoft database using the values set in the search record fields.

After you have set search criteria values in all the necessary search record fields with SetField, ProcessSearchDialog performs a database search for records matching the specified values. It is the equivalent of clicking the **OK** button in a search dialog box in the online system.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
-----------------	---

nMaxRows

The maximum number of rows to return. If you specify *nMaxRows* as 0, the Message Agent will return the row limit for your system (300 by default).

Return Value

Value	Meaning
PSMSG_OK	Success.
PSMSG_NOMATCHINGROWS	No database rows matched the search criteria.
PSMSG_NOTFOUND	The search criteria are not valid.
PSMSG_ERROR	General error.

SetField

Syntax

```
int stdcall PSMsgSetField(PSMSGHANDLE hContext, LPCSTR lpszFieldName, LPCSTR lpszValue);
```

Description

Sets the value of a message field to be mapped into a panel buffer when the message is processed.

The message definition contains a mapping between message input fields and panel record fields. To update a panel record field, your program passes message field names and values to the Message Agent. The Message Agent buffers these values until you use `ProcessMessage`. For more information about how the Message Agent applies these values, see `ProcessMessage`.

In addition to setting the value for a panel field, you can use `SetField` to constrain the values returned by `ProcessSearchDialog` or `ProcessPromptValue`. These functions will only return rows whose values match the partial value you set using `SetField`. In this use, the *lpszFieldName* parameter refers to the actual field name, not the field mapping name.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpszFieldName</i>	The name of the field to set a value for. The field must be designated as an input field in the message definition.
<i>lpszValue</i>	The value for the field.

Return Value

Value	Meaning
PSMSG_OK	Success.

PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMMsg function.
PSMSG_NOTFOUND	The requested field was not found.

SetOptions

Syntax

```
int stdcall PSMMsgSetOptions(PSMSGHANDLE hContext, int fOptions);
```

Description

Specifies whether to change from Add mode to Update/Display mode when the Message Agent finds a matching level 0 record.



This function is not available using the OLE Automation interface.

When the Message Agent is adding records, it may find an existing record that matches the record it's trying to add. When this happens, the Message Agent can:

- Report an error and not add the record.
- Switch to Update/Display mode and update the existing record with the new data.

You use PSMSG_OPT_CHANGEMODEADDUPDATE to specify which action the Message Agent takes.

This option applies only to records at level 0 on the panel. The message definition specifies how to handle matching records at other levels. Also, the option applies only to message definitions that use Add mode.

The PSMSG_OPT_NOGUI setting enables you to improve performance by suppressing the creation of the Message Agent interface controls. Since the Message Agent typically runs on an unattended server machine, it doesn't need to display its controls.

Parameters

hContext

The context handle assigned when this program connected to the Message Agent.

fOptions

One of the following keywords.

PSMSG_CHANGEMODEADDUPDATE (1) tells the Message Agent to switch to Update/Display mode if it finds a matching level 0 record while in Add mode. If you do not set this option, the Agent reports an error if it finds a matching record.

PSMSG_OPT_NOGUI (2) tells the Message Agent not to create or display the user interface controls on the panels it accesses. Setting this option overrides the setting from the registry.

PSMSG_OPT_NOFIELDEDITS (4) tells the Message Agent not to perform any validation edits on the incoming data.

PSMSG_OPT_RESET (0) returns all options to their defaults.

Return Value

<i>Value</i>	<i>Meaning</i>
PSMSG_OK	Success.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg.

StartMessage

Syntax

```
int stdcall PSMsgStartMessage(PSMSGHANDLE hContext,
LPCSTR lpszActivityName, LPCSTR lpszMessageDefn,
BOOL bOriginatorVerified);
```

Description

Notifies the Message Agent that the caller is ready to send and receive data for a defined message.

The StartMessage function tells the Message Agent which Application Designer message definition to use for mapping fields. You use this function before setting the input message fields with SetField or processing the message with ProcessMessage. You can start multiple messages during a single connection to a Message Agent.

If the Message Agent cannot open the message definition, the Message Agent goes into an error state until the next message is specified.

When someone creates a message definition in Application Designer, they have the option of requiring that the calling program verify the originator of the message, by means of an electronic signature or some other method. If a message requires originator verification but there is no way for the calling program to verify the originator, the Message Agent does not process the message.

Parameters

<i>hContext</i>	The context handle assigned when this program connected to the Message Agent.
<i>lpszActivityName</i>	Name of an activity created in the Application Designer.
<i>lpszMessageDefn</i>	Name of a message definition that is part of the specified activity.
<i>bOriginatorVerified</i>	Pass TRUE if your application has verified that the user who originated the message really sent it. An example is an email package that supports 'signing'. Pass FALSE if the verification failed, or if your application has no means of verifying the originator.

Return Value

Value	Meaning
PSMSG_OK	The Message Agent opened the message definition.
PSMSG_ERROR	General error.
PSMSG_NOTLOADED	Message Agent is not present.
PSMSG_BADCONTEXT	Invalid PSMSGHANDLE passed to PSMsg function.
PSMSG_NOTFOUND	The requested message definition was not found.

CHAPTER 11

Using Database Agents and Message Definitions

A **database agent** is a workflow program that performs a simple function: it runs a query against your PeopleSoft database and passes the results to the Message Agent.

Using database agents in conjunction with the Process Scheduler, you can monitor your database tables for conditions that should trigger business events. A database agent can run any query that you can define with PeopleSoft Query. The WHERE clause of the query is the business rule that's looking for exception conditions.

The Message Agent is a application server Tuxedo service that takes data from external applications and enters it onto a PeopleSoft panel. Essentially, it enables you to add automated steps to your business processes: the Message Agent enters the data instead of a flesh-and-blood user.

You create **message definitions** to tell the Message Agent how to apply the data it receives from the external application. A message definition associates a form or other electronic document with a PeopleSoft panel, setting up a correspondence between the fields in the electronic document and the record fields that underlay the panel.

Together, database agents and message definitions significantly extend the range of tasks you can automate. For example, you could create a database agent that queries for overdue receivables, and schedule it to run once a week. When the database agent finds overdue receivables, it passes the data to the Message Agent, which enters the data into a page that in turn generates work list entries for the appropriate AR clerks. All without user intervention.



For more information about developing and integrating Windows applications that use the Message Agent, see Message Agent.

Understanding Database Agents

Although database agents perform a simple task, they play an important role in workflow. By running predefined queries on a regular basis, they periodically check your PeopleSoft database for data that is relevant to your business processes. By passing the data to the Message Agent, they enable the system to respond automatically.

Monitoring the Database

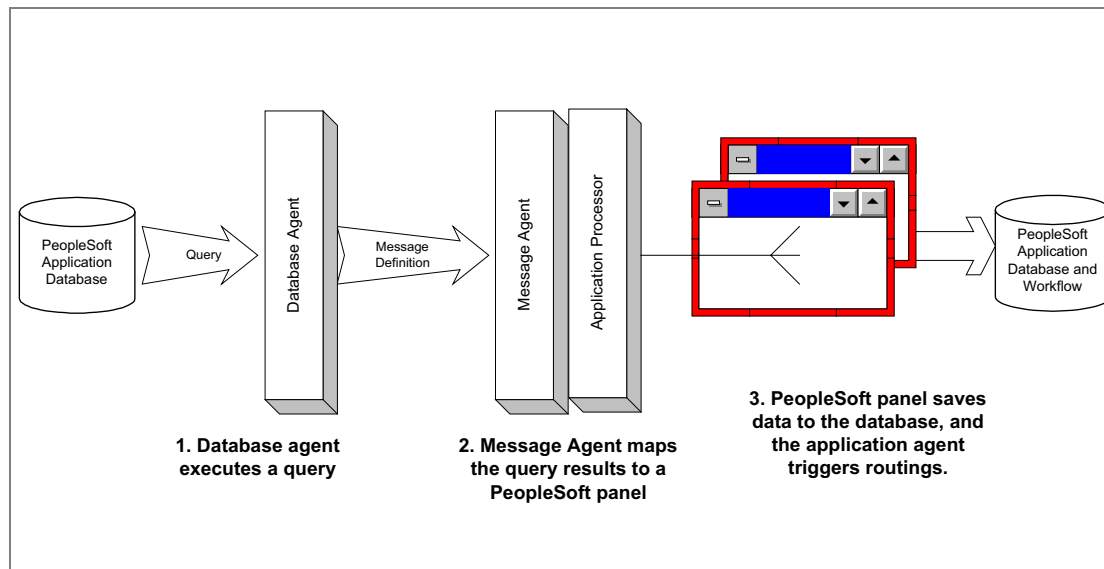
Most of the routings in a business process are triggered by a user entering data onto a page. For example, in a purchase order approval process, the system generates an approval request when a user enters a new purchase order. This immediate response is part of the productivity gain that PeopleSoft Workflow provides.

However, there are situations where you'd like to trigger routings based on some event that doesn't involve a user at all: the number of outstanding work items entries gets too high, a contract lapses, the Process Scheduler completes a batch process, or whatever. Database agents make such routings possible.

To trigger routings based on a non-user-initiated event, you use PeopleSoft Query to write a query that checks the database for data that needs to be processed. Using the Process Scheduler, you schedule a database agent to run the query on a regular schedule and kick off a workflow when the condition is true. Once you've started the database agent, no user intervention is required.



For an example of how we've used a database agent to monitor the PeopleSoft database, see Case Study: Remote Report Delivery.



Triggering Events through the Message Agent

Database agents do not trigger business events directly. They can't. With PeopleSoft workflow, you trigger business events by entering (and saving) data on a PeopleSoft page that has Workflow PeopleCode associated with it. Database agents can't enter data on pages; the only agents who can are flesh-and-blood users and the Message Agent.

Database agents trigger business events *indirectly*, by passing the results of their queries to the Message Agent. The Message Agent uses one of its message definitions to map the query results to a PeopleSoft page. If the page has Workflow PeopleCode associated with it, a business process is triggered when the Message Agent saves the page.

If a database agent query returns multiple rows of data—for example, if it finds several overdue items—the agent passes the rows to one or more Message Agents, one row at a time. If you have multiple Message Agents running, the database agent distributes the workload between them. The Message Agent invokes the component that triggers the business event once for each row it receives.



For more information about the Message Agent, see [Creating Message Definitions and Message Agent](#).

Creating a Batch of Online Processes

When you're working with PeopleSoft applications, there are some activities that you perform interactively (*online* processes) and some that you ask the system to perform behind the scenes (*batch* processes). Batch processes provide three major benefits:

- You can schedule them to run at a later time, on a recurring schedule if necessary.
- They can process a large number of items all together, unlike online processes which typically work on one item at a time.
- You can offload them to a server, so that time-consuming tasks don't tie up your machine.

However, batch processes have one possible drawback: they connect to the database directly, rather than working through the PeopleSoft pages. If you're counting on the Page Processor to validate incoming data or run some custom PeopleCode, you probably don't want a batch process updating the database behind its back. And since you trigger business events by saving data on a page, batch processes can't kick off a workflow.

You can use database agents to get around these limitations. Like online processes, database agents enter data through PeopleSoft pages (using the Message Agent as an intermediary); like batch processes, they can handle a batch of items. In essence, you can have an agent run a batch of online processes.

For example, suppose you have a batch process that transfers assets between departments, and you want to notify (via email) the managers of all affected departments. The batch process itself can't trigger an email routing, and to create the emails manually would be tedious and time-consuming. Instead, after the batch process is complete, you could run a database agent that queries for the transferred assets and sends an email to each manager. Or better still, you could replace the batch process with a database agent that makes the asset transfer. Since the database agent enters data through the normal asset transfer page, it can trigger an email routing as it makes the transfer.



One potential drawback to this approach may be performance. Most PeopleSoft batch processing uses “set” SQL processing; that is, they use SQL statements that process large volumes of data at once. Database agents process individual rows of data at a time.

Adding Database Agents to Your Workflow

The database agent program is simple, but fitting it into the workflow requires a good deal of planning.

In order for a database agent to trigger a business event, it has to pass its data to the Message Agent, and the Message Agent has to transfer that data to a page that has Workflow PeopleCode associated with it. So, in addition to the database agent itself, you have to make sure you’ve defined (1) a message definition that maps the query results to a page and (2) a page that triggers the appropriate business event.

To add a database agent (overview)

1. Identify the data you want the database agent to retrieve.

Before you create a database agent, you need to figure out what you’re looking for. If you’re creating an agent that monitors the database, you’re looking for data that tells you there’s work to be done: overdue employee reviews, inventory shortages, overdue accounts receivables.

2. Build a page that collects the information you need to start the process.

To trigger a business event, you need a page that can provide the system with the data it needs to route a work item to the next step in the workflow. The page must also use a record definition whose Workflow PeopleCode triggers the appropriate event. If you don’t already have an appropriate page, you need to build one.



For more information on setting up pages so they trigger a business event, see [Defining Event Triggers](#).

3. Write a query that retrieves the necessary information from the database.

From step 1, you know what condition you’re looking for, and from step 2, you know what data you’re going to need to trigger the business event. Use PeopleSoft Query to write a query that checks for the condition and retrieves the necessary data.



For more information about Query see [PeopleSoft Query](#).



To create a database agent query, your query profile must allow the creation of workflow queries; see Setting up Query Security.

Most of the time, your query will retrieve only key information from the database. When the Message Agent passes this key information to the page, the page retrieves the rest of the data for the item identified by the key.



Note: In PeopleSoft Query, make sure you identify the query as a database agent query using **File, Properties**, and that all headings in the query are specified as **Text** rather than **RFT Short** or **RFT Long**.

4. Design and build a business process for handling the data from the agent.

What processing steps need to occur when the database agent finds what it's looking for? Use the Application Designer to define a business process and the events that make it up.



For more information see Building Workflow Maps and Adding Events and Routings.

Make sure to include an activity with a database agent icon in it. When you complete the setup dialog boxes for the database agent icon, you define a message definition that tells the Message Agent how to transfer the data from the query result set to the page.



For more information, see Message Agent.

5. Test the database agent.

Run the database agent program from the Windows **Start** menu or create an icon. Check to see that you get the results you expect: that the database agent retrieved the correct data and that the correct business process was triggered.



For more information about the command line see Running Database Agents.

6. Add the database agent to the Process Scheduler.

Once you've verified that you're getting the correct results, add the database agent to the Process Scheduler. You specify what pages users run the database agent from and how often the system runs it.



For more information, see Adding Database Agents to the Process Scheduler.

7. Run the database agent.

Go to the page that you assigned the database agent to and click the **Run** button to run it.



For more information, see Starting Database Agents.

Running Database Agents

During testing, you can run the database agent program from the Microsoft Windows **File** menu or create a Windows icon for it. But when you go into production, you'll use the Process Scheduler to run the agent, even when you don't need it to run multiple times. See the next section to learn how to add agents to the Process Scheduler.

The command line to start a database agent is:

```
PSDBA -Aactivity_name -MDmessage_definition
[-TOPICtopic -T -Error_threshold -L -kbind1=bind_value1
-kbind2=bind_value2 ...]
```

You can use a hyphen or a forward slash (/) to introduce each parameter.

Don't include spaces between the command-line switches and their arguments. The arguments themselves can include spaces and don't need to be in quotes unless they contain a hyphen or slash. The order of the parameters doesn't matter.

<i>-Aactivity_name</i>	The name of the Application Designer activity that this database agent is part of.
<i>-MDmessage_definition</i>	The name of the message definition that defines the query and the page mapping for this database agent. The message definition has to be part of the specified activity.
<i>-TOPICtopic</i>	The topic name to use for communicating with the Message Agent. If you include this option, the database agent looks for (or starts, if you include the <i>-L</i> option) a Message Agent with the specified topic name.

If you don't include this option, the database agent uses the default topic name, PSMessagexx, where xx is a two-digit number. It looks for any available Message Agents with such topic names and distributes its result set among them. If you include the -L option, the database agent starts the number of instances specified using the Message Agent Server setting in the Configuration Manager, and distributes its result set among them.

-T	Turns on the trace file option. When you include -T on the command line, the database agent creates a trace file in your Windows TEMP directory. Its filename is DBA*.TMP, where * is a unique alphanumeric character.
-Error_threshold	The number of errors the database agent tolerates before aborting. The default value is 1. When you specify a number higher than one, the trace file only contains the text of the last error encountered.
-L	Instructs the database agent to start the Message Agent before executing its query, then close the Message Agent when it's done. If you don't specify a topic name using the -TOPIC parameter, the database agent starts the number of instances you specified using the Message Agent Server setting in the Configuration Manager; in this situation, the Message Agents don't close after the database agent disconnects.
-kbind1=bind_value1	A value for the first bind variable in the query that the agent runs. If the query doesn't have bind variables, don't include any -k arguments. If the query includes multiple bind variables, you can include multiple -k arguments.



In addition to the parameters listed here, the PSDBA command line can include any of the login parameters that are available for PeopleTools executables.



For more information see PeopleTools Command Line Parameters.

Adding Database Agents to the Process Scheduler

The steps for scheduling a database agent are the same as those for any Process Scheduler process.

- Define a process type definition. The process type definition specifies the general parameters

that all database agents share.

PeopleSoft ships with the process types for database agents already defined, so you can skip step 1. We recommend that you always include the /L option in the process type definitions for database agents.

- Create a process definition. When you add a database agent to the Process Scheduler, you provide the unique information about this agent. Most importantly, you specify what activity name and message definition define the database agent and how often the Process Scheduler runs it.



For more information about any of the Process Scheduler options, see Process Scheduler Development.



The default process type definitions for database agents use the %INSTANCE% variable for the -TOPIC parameter. This setting ensures that each database agent starts its own instance of the Message Agent. If you want to take advantage of the Message Agent Server functionality—where the database agent starts multiple instances of the Message Agent and distributes its work among them—remove the -TOPIC setting from the process type definitions.

Database agents can only run on machines running Windows 95 or Windows NT. That's because it needs to communicate with the Message Agent and PeopleSoft applications.

To add a database agent to the Process Scheduler

1. Select PeopleTools, Process Scheduler Manager, Use, Process Definitions.
2. Search for an existing process or add a new one.

Use the standard search or add method to enter the **Process Type** and **Name**. Be sure the **Process Type** is *Database Agent*.

The **Process Definitions** page appears.

Process Definition | Process Definition Options | Override Options | Destination

Process Type: Database Agent
Name: ROUTEPR

***Description:** Route Purchase Requests
Long Description: This database agent is used in the Purchase Request example. It routes approved purchase requests to buyers.
***Priority:** Medium

☒ API Aware
☒ Log client request
☐ SQR Runtime

Save Return to Search Previous tab Next tab Add Update/Display

Process Definition | Process Definition Options | Override Options | Destination | Page Transfer | Notification

Process Definitions page

In this component, you tell the Process Scheduler where, how, and how often to run the database agent.

3. Enter a short description of the database agent in the **Description** text box and a more detailed description in the **Long Description** text box.
4. Select the **Log client request** and **API Aware** check boxes.

These check boxes tell the Process Scheduler that the database agent program can and will update the Process Monitor status tables.

5. In the **Priority** list box, specify what priority the Process Scheduler should give to this process.

The priority determines the order in which the Process Scheduler runs processes when more than one is scheduled for the same time.

6. Move to the Process Definition Options page.

Process Definition Options page

7. In the **Run Location** field, select **Server**.

If you want this database agent to always run on the same server, enter the **Server Name**; if you don't care which server it runs on, leave the **Server Name** blank.



In a production environment, you'll usually want to have a server dedicated to running database agents.

8. In the **Recurrence Name** list box, specify how often you want the Process Scheduler to run this database agent.

If none of the available values give you the option you want, don't worry. You'll be able to add a new recurrence name when you run the database agent for the first time.

9. In the **Component** box, select the components containing the pages from which user will be able to run the database agent.

If you don't have a special page in mind (and the query doesn't include any bind variables), you can use RUN_DBAGENT from the Workflow Administrator.



For more information about running database agents from the selected page, see Starting Database Agents.



To add additional pages, click **Add** to add rows in the **Component** box.

10. In the **Process Security Groups** box, select the security groups that have the right to run this database agent.

If there's more than one security group, click **Add** to add additional groups.



For more information about process security groups, see Understanding PeopleSoft Security.

11. Move to the next page in this component.

The Override Options page appears.

Override Options page

This page is where you specify the name of the query the database agent runs. If the query includes runtime bind variables, you provide values for them too.

12. Select the **Append** option to add parameters to the **Parameter List**.

13. Enter these command line parameters in the right text box:

```
-Activity_name -MDmessage_definition [-T -Error_threshold
-kbind1=bind_value1 ...]
```



For more information about these command line parameters, see Running Database Agents.

The activity name and message definition name are required for all database agents. The bind variable arguments are required if the specified query has bind variables. The bind variable values must be either constants or fields from the run control record.

14. Save your work.

You don't need to add anything to the remaining pages in this component.

Starting Database Agents

Adding a database agent to the Process Scheduler makes it available, but it doesn't actually schedule the agent to run. You start a database agent just like you start other scheduled processes, using the **Process Scheduler Request** dialog box.

Assigning Database Agents to Components

When you add a database agent (or any other scheduled process, for that matter) to the Process Scheduler, you specify which component users can start it from. When users select Run from that component, the Process Scheduler Request dialog box lists the database agent as one of the processes they can request.

To schedule a process, you pass the Process Scheduler a **run control**. So, the component you assign the database agent to must be one that creates run control records.



For more information about run controls, see Process Scheduler.

If the query that your database agent runs doesn't include any runtime bind variables, you can assign the database agent to an existing component that creates run controls. The system gets all the run control information it needs from the **Process Scheduler Request** dialog box. We recommend that you assign these database agents to the RUN_DBAGENT component, which appears in the Workflow Administrator under the menu item **Process, Database Agent**.

If the query does include runtime bind variables, you need to create a run control record definition that includes the necessary fields, build a page on which the user can enter values for the fields, and assign the database agent to this page. For example, if the database agent queries for all employees in a specified department, you'll need a run control record that includes a DEPTID field and a page with an entry field for DEPTID.

To schedule a database agent to run

1. Navigate to the component that you assigned the database agent to.

For example, if you assigned it to the RUN_DBAGENT component, select **PeopleTools, Workflow Administrator, Process, Database Agent**.

2. Search for an existing **Run Control ID** or add a new one.

Use the standard search or add method to enter your Message ID and access the first page in the component.

3. If the database agent query includes bind variables, enter values for them on the page that appears.

If you're on the Database Agent page in the Workflow Administrator, there are no editable controls.

4. Click in the toolbar or select **File, Run**.

This option displays the **Process Scheduler Request** dialog box so that you can specify when, where, and how often to run the database agent. For details about the **Process Scheduler Request** dialog box, see Process Scheduler.

Case Study: Remote Report Delivery

This section describes one of the database agents we deliver with PeopleSoft applications. We hope that giving examples of how we've used a database agent will help you implement your own.

The Remote Report Delivery feature enables users to request reports by means of an email message or form, and to receive the report back via email. The system uses a database agent to check for completed reports that it needs to deliver back to the requester.

First, a quick overview of how the feature works. A user sends a specially formatted message to a mailbox that the PeopleSoft system is monitoring. Using the Message Agent, the system adds the report request to the Process Scheduler. The Process Scheduler prints the report and places it in a particular directory. A database agent queries for completed reports, and when it finds some, sends an email message to the requester with the report attached.

The Query

The Remote Report Delivery database agent runs the query [DBAG] Report Delivery. The SQL for this query is:

```
SELECT A.REPORT_INSTANCE
FROM PS_RPT_DLVRV_VW A
WHERE A.RPT_DELIVERY_STAT = 'P'
```

RPT_DLVRV_VW is a view that includes the completed reports from the report request table (RPT_RQST). This query selects the Instance ID for all reports that haven't been delivered yet (whose status is 'P' for "Pending").

The Message Definition

The message definition for the [DBAG] Report Delivery query maps the Instance ID to the REPORT_DELIVERY component in the Workflow Administrator. This component has fields for all the details about the report request: who requested it, what run control it used, and so on.

The message definition specifies that the Message Agent accesses the component in Update/Display mode. Just as it would if a user selected Update/Display mode, the system presents a search dialog box. The Message Agent maps the Instance ID to the search key field, and the system retrieves the report request details and displays the component. The Message Agent saves the component, which causes the system to run all the PeopleCode associated with the component.

The Component

The sole purpose of the REPORT_DELIVERY component is to retrieve report details and trigger an email back to the report requester. The record definitions that underlay this component include PeopleCode functions that perform two crucial tasks:

- The RPT_RQST_VW record definition changes the report delivery status from 'P' (Pending) to 'D' (Delivered)
- The RPT_RQST_WRK record definition triggers the Report Delivery business event.

For the Message Agent to be able to use the component, we had to assign it to a menu using the Application Designer. We added it to the Workflow Administrator. However, since *only* the Message Agent uses this component, we didn't give users access to the component in Security Administrator.

Troubleshooting the Database Agent

The Database Agent, a Windows application, has not changed much in the new Message Agent architecture. Its purpose is to retrieve the results of a query and issue commands via the Message Agent to insert/update data into PeopleSoft components. The only requirement we must impose is that the Database Agent has to run in three-tier mode. That is, the command line generated to launch the Database Agent, whether on a PeopleSoft Windows client or an NT Process Scheduler server, must have signon parameters that will allow you to connect to an Application Server. We have modified the Process Type definitions for the Database Agent to reflect that rule. If you attempt to launch a Database Agent process in two-tier mode through the Process Scheduler, your process will not run. /CX and /TOPIC parameters are now required unlike previous versions of PeopleTools.

In addition to SQL and PeopleCode traces, you can view the history of a Database Agent process in another trace file. To invoke the Database Agent trace, you'll need to append a /T parameter to the command line. It will create a DBA*.tmp file on the client's %TEMP% directory. Within it, you can see which Activity, Message Definition, Query and Application Server are being used. You'll see the results of the query listed in the trace file and each individual Message Agent transaction being performed.

```
/* results from query */
JOB_REQ#=000000
Job Code=1051
Department=10300
Position=00000008
```

```
/* results from performing the Message Agent transaction */
```

```
11.42.28 : StartMessage
JOB_REQ#=000000
Job Code=G061
Department=10200
Position=00000008
: ProcessMessage
```

The above shows an example of a successful transaction. If the transaction had failed, the ProcessMessage line would not appear and be replaced by an error message. The /E parameter will determine how many of those errors the Database Agent can encounter before stopping the process altogether.

If you attempt to launch a Database Agent process in two-tier mode, the trace file will typically have a comment in there saying that it's missing a value for the /TOPIC parameter. Most errors will generate a message "Data buffer error, verify query field mapping" to appear in your Process Monitor. This message implies that at least one of the Message Agent transactions was not successfully processed. To isolate the problem, you may want to get your hands on a utility program, like the Message Test program, that will allow you to feed the same offending data values over and over again to the Message Agent to help troubleshoot the problem.



For more information see the Troubleshooting the Message Agent.

Understanding the Message Agent and Message Definitions

Message Agent

The Message Agent is a Tuxedo service, residing on the application server, that accepts messages from Windows applications. The messages allow third-party applications to enter data into, and get data out of, PeopleSoft applications. PeopleSoft performs all the same edits and security checks it always does, including running any PeopleCode associated with the page. So, if the page has Workflow PeopleCode associated with it, the Message Agent can trigger a business event.

The Message Agent extends the reach of your business processes beyond the users who use PeopleSoft applications. Rather than typing data directly into a PeopleSoft page, people throughout the organization can enter the data using software they're familiar with.



The primary use of the Message Agent is entering data *into* the PeopleSoft database. However, you can also develop applications that use the Message Agent to retrieve data *from* the PeopleSoft database. For more information see Message Agent.

Message Definitions

To allow other programs to be able to “fill in” a PeopleSoft page, you create a *message definition* in the Application Designer. The definition creates a mapping between the fields passed to the Message Agent and the fields on a PeopleSoft page. It specifies:

- **A unique name.** When a Windows application sends a message to a Message Agent, it tells the Agent which message definition to use.
- **The page to enter data into, the action mode, and the search record definition.** When a Message Agent starts using a message definition, it navigates to the corresponding PeopleSoft page. It gets to the page just as a user would: by selecting the page name from a menu, selecting an action mode from the cascading menu, and entering key data into a search dialog box.
- **Which data fields to map to which page fields.** The Message Agent gets the values from specified fields and enters them into the corresponding fields on the PeopleSoft page. The Message Agent can also copy values from the page back to the external application, so the application can verify the results or pass them on to the user.
- **How to process fields controlled by scrolls.** A page that has scroll bars can include multiple rows of data from the database. The message definition specifies whether the Message Agent can add rows to the scroll, update existing rows, or delete rows.
- How to respond to the user.

Creating Message Definitions

The Message Agent accepts data from a variety of sources, such as electronic forms software, interactive voice response (IVR) systems, or World Wide Web applications. The toolbar for creating activities in Application Designer includes several icons that represent these application types.









To create a message definition, you select the icon that best represents the external application the Message Agent is getting data from. You add the icon to the activity the external application is part of, then create a mapping between the fields of data provided by the external application and fields in PeopleSoft record definitions.

You create one message definition for each form, query, or electronic document type you want the Message Agent to process.

To create a message definition

1. Open the activity.

The activity toolbar appears. Several of the icons represent types of external applications that can send data to the Message Agent.

Button	Description
	Inbound form
	Interactive Voice Response.
	Web link.
	Kiosk
	EDI
	Other external application.
	Database agent.
	NVision



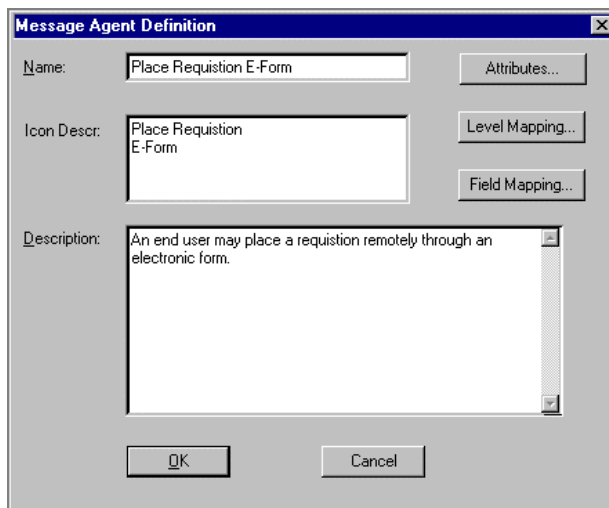
For information about opening or creating activities, see Building Workflow Maps.

2. Add to the map the icon corresponding to the source of the data.

Click the icon in the toolbar, then click on the map where you want it to appear.

3. Right-click the icon on the map and select **Component Properties**.

The **Message Agent Definition** dialog box appears.



The dialog box titled "Message Agent Definition" contains the following fields and buttons:

- Name:** A text box containing "Place Requisition E-Form". To its right is an "Attributes..." button.
- Icon Descr:** A text box containing "Place Requisition E-Form". To its right are "Level Mapping..." and "Field Mapping..." buttons.
- Description:** A large text area containing "An end user may place a requisition remotely through an electronic form." Below this area are "OK" and "Cancel" buttons.

Message Agent Definition dialog box

4. Enter a name and description.

The text you enter in the **Name** text box is the name that the external application uses to identify this message definition. It must be unique.

By default, the name also appears as the display text under the icon on the map. If you want different text to display under the icon, enter it in the **Icon Descr** text box. You can include line breaks in the text by pressing the ENTER key; these line breaks will appear in the icon display text.

You can enter a more detailed explanation of the message definition in the **Description** text box. We recommend that you describe what application uses this message definition and what it accomplishes.

5. Click the **Attributes** button.

The **Message Attributes** dialog box appears.

The screenshot shows the 'Message Attributes' dialog box. It has a title bar with a close button. Inside, there's a section labeled 'Target Panel Group' containing five dropdown menus. The values are: Menu Name: PURCHASE_REQUEST, Bar Name: USE, Item Name: CREATE_PURCHASE_REQUEST, Action Name: &Add, and Search Record: WFEXPR_HDR. At the bottom, there are two buttons: OK and Cancel.

Message Attributes dialog box

If you're creating a message definition for electronic forms, you'll see the **Message Agent Form Attributes** dialog box instead of the **Message Attributes** dialog box.

The screenshot shows the 'Message Agent Form Attributes' dialog box. It has a title bar with a close button. Inside, there's a section labeled 'Target Panel Group' with the same five dropdown menus as the previous dialog. To the right of this is a section labeled 'Forms Attributes'. Within this section, there's a 'Reply Method' group with three radio buttons: None, Reply, and Forward. Below these are two rows of radio buttons for 'When OK' and 'When Error'. At the bottom of the 'Forms Attributes' section is a checkbox labeled 'Verify Originator Id'. At the bottom of the dialog are two buttons: OK and Cancel.

Message Agent Form Attributes dialog box

You use either of these dialog boxes to specify which page the Message Agent enters data into. The **Message Agent Form Attributes** dialog box also contains some settings that are relevant for forms only.

6. Choose the target component.

Use the first three list boxes in the **Target Component** box to identify the page the Message Agent should enter data into. In the **Action Name** list box, specify what type of action the Message Agent needs to perform on the database; the list includes only those actions that are available for the selected page.

7. Choose the search record for the component.

In the **Search Record** list box, select the record definition to use for locating the appropriate record. The Message Agent uses this record definition rather than the standard search record or add mode search record specified for the page when you created the component. The record definition you select must include all the level 0 key fields for the selected page.



For more information about associating search records with components, see [Creating Component Definitions](#).

8. Specify a query to run (database agents only).

A database agent runs a query against your PeopleSoft database and passes the results to the Message Agent. In the **Query** list box, select the query that the database agent runs.



For more information about database agents, see [Adding Database Agents to Your Workflow](#).

9. Specify a reply method (electronic forms only).

The reply method specifies what kind of feedback you want to provide to users who send forms to the Message Agent. The options are:

None	The user doesn't get any acknowledgment.
Reply	The user gets an email message confirming the processing of the form or giving the text of any error message.
Forward	The user gets a copy of the form returned to him or her, filled in with the field values that the Message Agent set. If there was an error, the Message Agent returns the field values as the user entered them.

You can specify a different reply method for when the Message Agent successfully maps the form fields to the page (**When OK**) and when an error occurs (**When Error**).

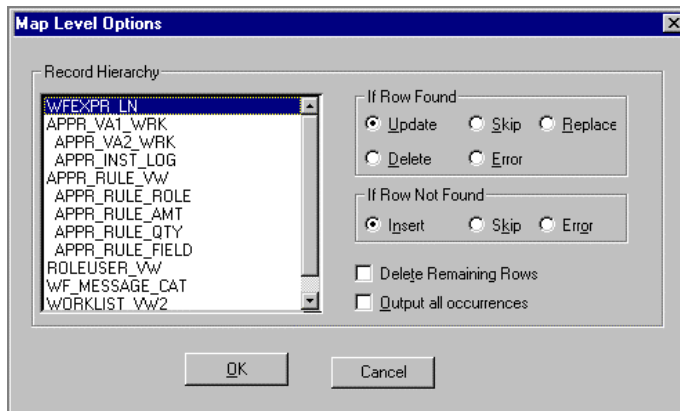


Some forms products, such as Lotus Notes, have an electronic signature feature that enables it to verify the identity of a person who mails a form. If you select the **Verify Originator Id** option, the Message Agent only accepts forms that were signed by a verified user.

10. Click the **OK** button to close the dialog box.

11. Click the **Level Mapping** button.

The **Map Level Options** dialog box appears.



Map Level Options dialog box

You use this dialog box to specify how the Message Agent handles fields controlled by a scroll bar; that is, fields at levels higher than 0. The **Record Hierarchy** box lists the record definitions associated with scroll bars in the page you are mapping to.

If the page doesn't have any scroll bars, the list is empty. However, you still need to select one of the options in the **If Row Found** and **If Row Not Found** boxes.

12. Set the map level options.

You can set different map level options for each scroll on the page. Highlight a record definition in the **Record Hierarchy** box before setting the options. If the box lists more than one record definition, set the options for each one in turn.

When the Message Agent is ready to map the key fields for a scroll, it checks whether a row with those key values already exists. If one does, the Message Agent takes the action you select in the **If Row Found** box. If no such row exists, it takes the action you select in the **If Row Not Found** box. The table below describes the available actions.

For level 0 fields, the Message Agent adds or updates records based on the **Action Name** you selected for the page.

Option	Action
Update	Replace the values in the existing row with the values from the external application. Leave the values of any unmapped fields

Option	Action
	unchanged.
Delete	Remove the existing row and don't insert a new one.
Skip	Discard the values from the external application, and don't do anything to the existing row (if there is one).
Error	Report an error, and don't do anything to the existing row (if there is one).
Replace	Delete the existing row and insert a new row using the values from the external application. Replace differs from Update in how it handles fields for which the external application doesn't provide values. Update leaves the fields with their values from the existing row; Replace returns them to their default values.
Insert	Add a row to the scroll using the values from the form or query. The external application must provide values for all required fields.

13. Click the **Delete Remaining Rows** check box if you want the rows that the Message Agent maps to replace any existing rows, rather than adding to them.

When this check box is selected, the Message Agent removes any rows in the scroll that the external application does not provide values for. If the check box is not selected, the Message Agent adds or updates the rows you provide values for and leaves other rows in the scroll unchanged.

14. Set the Output all occurrences option.

The **Output all occurrences** check box specifies how the Message Agent handles output fields that are inside a scroll.

Output fields are record fields whose value the Message Agent copies *from* the record *to* the corresponding form field. If an output field is inside a scroll, there may be multiple values for that field—one for each row in the scroll. If you select the **Output all occurrences** check box, the Message Agent makes all the values for the field available; if you don't select it, the Message Agent returns only the value from the first row.

An example of this would be to return all of the rows of training classes that a person is enrolled in.



This option is useful only if the external application is capable of processing multiple rows, and is *available for level 1 scrolls only*.

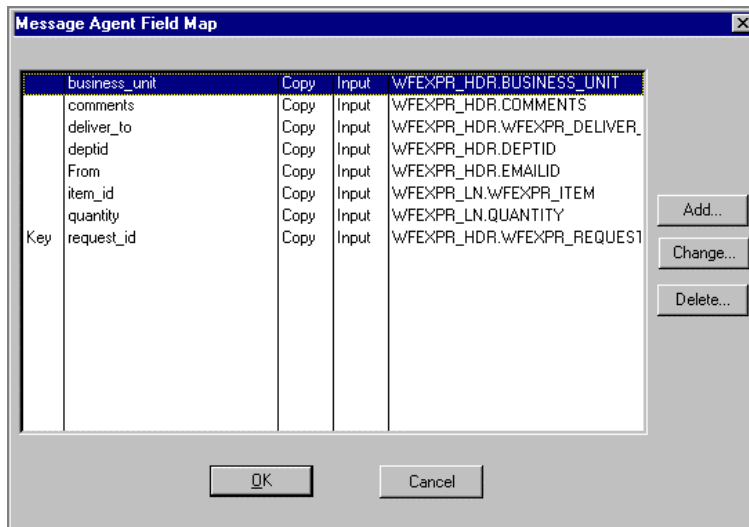


For information on writing an external application that retrieves multiple rows of output data, see Message Agent.

15. Click the **OK** button to close the dialog box.

16. Click the **Field Mapping** button.

The **Message Agent Field Map** dialog box appears.



Message Agent Field Map dialog box

This “map” shows the relationship between the fields from the external program and PeopleSoft record fields.

When you first open the dialog box for a new message definition, the list is blank unless you’re creating the message definition for a database agent. When you’re mapping a database agent query, the Message Map displays the column headings from PeopleSoft Query. It displays the headings rather than the field names because a query can include fields with the same name (from different record definitions), but the column headings should let you differentiate them.



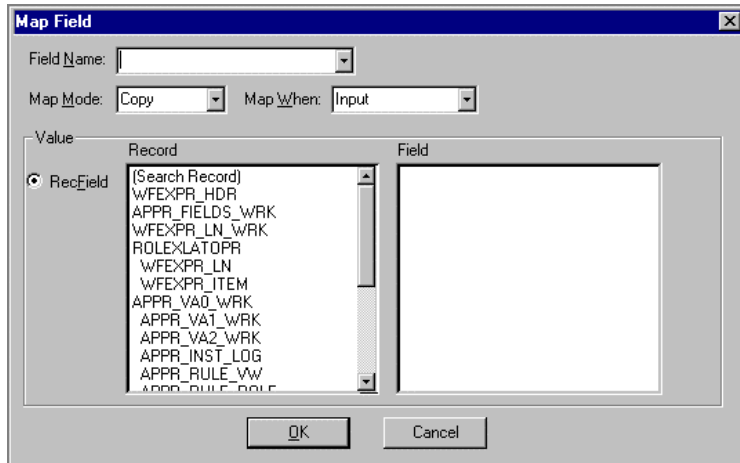
Note: The order in which you map fields is critical. You have to map values into the PeopleSoft record fields in a specific order:

- a) You must map the key fields before any other fields at the same level.
- b) If the page you’re mapping to has multiple levels, you must map the fields from level 0 before level 1, level 1 before level 2, and so on.

17. Click the **Add** button to add a new field mapping.

Or, to modify an existing field mapping, double-click it in the Message Map, or highlight it and click the **Change** button.

The **Map Field** dialog box appears.



Map Field dialog box

You use the **Map Field** dialog box to specify the relationship between a field of data from the external program and a PeopleSoft record field.

18. Specify the field to map.

If you selected a field name in the Message Map dialog box, the **Field Name** list box should show the field. If you're adding a new field to the map, type its name in the box or select it from the list.

19. Set a value for the field.

The **Record** box lists the record definitions that are available for mapping, namely the **(Search Record)** and the record definitions associated with the component for this message definition. Highlight the record that contains the field you want; the record's fields appear in the **Fields** list box. Highlight the one you want to link to the selected external field.

20. Set the Map Mode and Map When options.



Most of the time you'll leave the **Map Mode** and **Map When** list boxes with their default values: **Copy** and **Input** respectively. The other values are useful only if the external application accepts output values from the Message Agent.

Although the primary purpose of a message definition is to add data to PeopleSoft, the Message Agent can actually establish two-way communication with the external application. After the Agent transfers data to the page, it builds a list of output fields and makes their values available to the external application.

So, when you map each field, you'll identify it as an input field, output field, or both:

- An **input field** is a form field whose value the Message Agent copies from the form to the corresponding record field.
- An **output field** is a record field whose value the Message Agent copies from the record to the corresponding form field after mapping all input fields.

In the **Map When** list box, specify whether the Message Agent should copy the value *from* the form field to the record field (**Input**), from the record field *to* the form field (**Output**), or **Both**.

You would use **Both** when the form is multi-purpose for input and output functions.



For more information about establishing two-way communication with an external application, see Message Agent.

The value in the **Map Mode** list box applies only to output fields. It tells the system whether to copy the assigned value directly into the output field or to replace the assigned value with the corresponding value from the Translate Table. For example, suppose you associate an Employment Status field with the record field PERSONAL_DATA.STATUS. If the PERSONAL_DATA.STATUS field holds an abbreviation that translates to a value on the Translate Table, you can tell the system to fill in the output field with the abbreviation (**Copy**) or with the corresponding Translate Table value (**Xlat-S** for the short form or **Xlat-L** for the long form).



The **Xlat-S** and **Xlat-L** options only appear if the selected record field has Translate Table values associated with it.

21. Click the **OK** button to add the field association to the Message Map.

The record field you selected appears in the **Value** column.

22. Highlight the next field you want to map, or click the **Add** button, and repeat steps 18 to 21.

Creating a Valid Message Map

To create a valid message map, you must map an input field to every required field in the record definitions, including all key fields. Furthermore, each external application that uses this message definition must provide values for those fields.



The Message Agent can't map data into fields that have default processing associated with them. For example, it can't map into a Request ID field that automatically fills in the next available Request ID. If the search dialog box for a page has default processing, skip that field in the message map.

Handling File Attachments

The Message Agent can't transfer a file attachment to a PeopleSoft page field. If an incoming form has a file attachment, the program that passes it to the Message Agent has to handle it somehow. For example, the program that we deliver for Lotus Notes integration, PSNOTES.EXE, detaches any attached file and places it in the directory specified by the DetachDir setting in the Microsoft Windows registry. It gives the file a unique name using a consecutive numbering scheme.

When PSNOTES.EXE detaches a file attachment, it writes the name and location of the file in the FileName and FilePath fields. If you want to store the filename and location in the PeopleSoft database, map these two fields to record fields.

Mapping to Pages with Scrolls

If the page for this message definition includes a level 1 scroll, a single transaction can provide values for more than one row in the scroll. To provide values for more than one row, map more than one field to the same level 1 record field. For example, a form for specifying beneficiaries could have fields named BENEFIC1 and BENEFIC2, both of which you would map to the level 1 record field BENEFICIARY. When the Message Agent maps these fields, it will create two rows in the scroll, one for each value in the form.



You can provide values for multiple rows only for level 1 scrolls. For level 2 and level 3 scrolls, you must process one row at a time.



For more information about managing multiple scroll levels with the Message Agent, see Message Agent.

1. Click the **OK** button in both dialog boxes to close them.
2. Connect the icon to the business event or step that follows it in the activity.

If the data that the Message Agent enters on the page triggers a business event, add the business event icon to the activity and draw a connecting arrow from the incoming data icon to the event icon.

If the next step in the activity is another page, connect the incoming data icon to the following step icon.



For more information about adding icons to activities and connecting icons within activities, see Building Workflow Maps.

CHAPTER 12

Outgoing Forms API

The PeopleSoft Application Designer enables application developers to implement *routings*, which transfer data from one step in a business process to another. One of the available types of routings is a *forms routing*. With a forms routing, the system takes data from a PeopleSoft panel the user is working on, enters it onto a third-party form, and mails the completed form to designated users by means of the forms product's mail capabilities.

This chapter describes the programming requirements for integrating an electronic forms package with PeopleSoft applications so that it can accept forms routings. It includes complete reference material for the PeopleSoft Forms API.



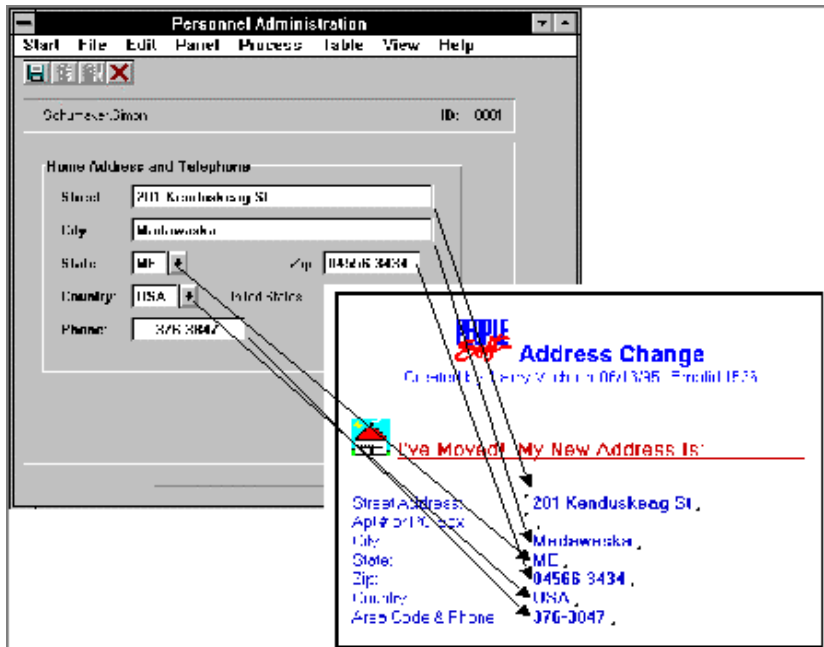
You can also integrate electronic forms software so that users can send forms to PeopleSoft applications using the Message Agent.



For more information about the Message Agent, see the Message Agent documentation.

Understanding Forms Routings

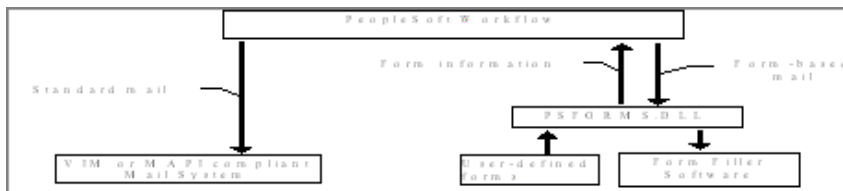
To design a forms routing, a developer maps the fields in a PeopleSoft panel to the fields in the form it wants to generate. When a user triggers the forms routing, the system copies the data from the panel fields to the corresponding form fields, then sends the form.



Mapping an Outgoing Form

PSFORMS.DLL

PeopleSoft applications communicate with the forms software by means of a DLL named PSFORMS.DLL. To integrate a forms product with PeopleSoft applications, you create a version of PSFORMS.DLL that supports the function calls the PeopleSoft application uses to generate and send forms.



Communicating with the Forms Software

The function descriptions in this section describe the function calls that PeopleSoft applications make to PSFORMS.DLL and the responses it expects to the calls.

Since you'll want PSFORMS.DLL to work at any installation, you don't want to build in any site-specific information, like the name of the forms server and database that the PeopleSoft applications accesses. For site-specific information, you can add entries to the Microsoft Windows registry. PSFORMS.DLL can read the entries to identify the forms server, database, and any other site-specific information it needs.



We concentrate on the right side of the diagram above. PeopleSoft 7 supports any VIM- or MAPI-compliant email package. No further development is necessary.

Forms API

Operations by Functional Category

Session Level Operations

PsfGetAPIInfo	Provides the name of forms product this DLL supports and returns the version number of the API it implements.
PsfOpenSession	Connects the PeopleSoft system to the forms interface DLL using a specified login name and password.
PsfCloseSession	Closes a session.

Query Operations

PsfGetFormCount	Counts the available forms.
PsfGetFormList	Lists the available forms.
PsfGetFieldCount	Counts the fields on a form.
PsfGetFieldList	Provides a list of the fields on a form.
PsfGetLastError	Gets the error message text associated with an error code.

Send Operation

PsfSendForm	Sends a completed form through the mail system.
-------------	---

PsfCloseSession

Syntax

```
int FAR PASCAL PsfCloseSession(HSESSION hSession);
```

Description

Closes a session.

Parameters

hSession The session handle assigned by PsfOpenSession when PeopleSoft connected to PSFORMS.DLL.

Return Values

Value	Code	Meaning
PSF_OK	0	The session was successfully closed.
PSF_NOSESS	4	<i>hsession</i> doesn't identify a current session.

PsfGetAPIInfo

Syntax

```
int FAR PASCAL PsfGetAPIInfo(LPPSFDEFNKEY lpDefKey);
```

Description

Provides the name of forms product this DLL supports and returns the API version.

PeopleSoft uses the same programming interface for all forms products. It uses the PsGetAPIInfo function to get the product name of the forms software your version of PSFORMS.DLL supports.

PsfGetAPIInfo puts the product name as a string in the name field of the structure.

Parameters

lpDefKey A pointer to a structure containing a character array for the function to write to. The form of the structure is:

```
typedef struct
{
    char    name[PSF_FIELDNAMELEN + 1];
} PSFDEFNKEY;
```

Return Value

An integer identifying the version of this API the DLL implements. The value is set in the PeopleSoft forms interface header file.

PsfGetFieldCount

Syntax

```
int FAR PASCAL PsfGetFieldCount(HSESSION hSession, LPPSFDEFN lpFormData);
```

Description

Counts the fields on a form.

PsfGetFieldCount gives the number of fields in a form by entering an integer in the wNumFields field of the *lpFormData* structure. The name of the desired form is in the structure's formName field.

The PeopleSoft system uses the field count to allocate an array large enough to hold all the fields. It passes a pointer to this array to PsfGetFieldList.

Parameters

<i>hSession</i>	The session handle assigned by PsfOpenSession.
<i>lpFormData</i>	A pointer to a structure that contains form information. The form of the structure is: <pre>typedef struct { PSFDEFNKEY formName; // Name of the form WORD wNumFields; // Number of fields char cSep; // Separator character LPPSFIELDVAL lpFields; // Pointer to field array } PSFDEFN;</pre>

Return Values

Value	Code	Meaning
PSF_OK	0	The function executed successfully.
PSF_NOMAIL	2	Mail system failure.
PSF_NOFORM	3	The specified form is not accessible.
PSF_NODB	5	The forms database is not accessible.

PsfGetFieldList

Syntax

```
int FAR PASCAL PsfGetFieldList(HSESSION hSession, LPPSFDEFN lpFormData);
```

Description

Provides a list of the fields on a form.

PsfGetFieldList provides the PeopleSoft application with information about the fields in a form. The name of the desired form is in the structure's formName field. The structure's lpFields field points to the array of structures that PsfGetFieldList updates. Each item in the array has this structure:

```
typedef struct
{
    char    szFldName[PSF_FORMNAMELEN + 1];    // Field name
    WORD    wFldSize;                          // Field size
    LPSTR    lpszStrVal;                       // Field value
} PSFFIELDVAL;
```

PsfGetFieldList enters data into the szFldName and wFldSize fields. The PeopleSoft system will provide the field values before passing the structure to PsfSendForm.

PsfGetFieldList includes all fields required to send forms using the forms product, regardless of whether they appear on the user's form. For example, if a user's form contains a TO field but no CC field, PsfGetFieldList includes both fields since a CC field is required to transmit a mail message. If the forms mail system supports and requires attributes such as sensitivity, return receipt, and priority, PsfGetFieldList also returns these items as fields.

Parameters

hSession

The session handle assigned by PsfOpenSession when PeopleSoft connected to PSFORMS.DLL.

lpFormData

A pointer to a structure that contains form information. The form is:

```
typedef struct
{
    PSFDEFNKEY    formName;    // Name of the form
    WORD          wNumFields;  // Number of fields
    char          cSep;        // Separator
character
    LPPSFFIELDVAL lpFields;    // Pointer to array
} PSFFORMDEFN;
```

Return Values

Value	Code	Meaning
PSF_OK	0	The function executed successfully.
PSF_NOMAIL	2	Mail system failure.
PSF_NOFORM	3	The form is not accessible.
PSF_NODB	5	The forms database is not accessible.

PsfGetFormCount

Syntax

```
int FAR PASCAL PsfGetFormCount(HSESSION hSession, LPINT lpnCount);
```

Description

Counts the available forms.

PsfGetFormCount sets *lpnCount* to a pointer to the number of forms available in the forms database. The PeopleSoft system uses the count to allocate an array large enough to hold all the forms. It passes a pointer to this array to PsfGetFormList.

Parameters

hSession The session handle assigned by PsfOpenSession when PeopleSoft connected to PSFORMS.DLL.

lpnCount A pointer to an integer that will receive the count.

Return Values

Value	Code	Meaning
PSF_OK	0	The function executed successfully; <i>lpnCount</i> is set to the number of forms.
PSF_NOMAIL	2	Mail system failure.
PSF_NODB	5	The forms database is not accessible.

PsfGetFormList

Syntax

```
int FAR PASCAL PsfGetFormList(HSESSION hSession, LPPFSFORMLIST lpFormList);
```

Description

Lists the available forms.

PsfGetFormList provides a list of the available forms in the forms database. The structure's lpForms field points to the array of structures that PsfGetFieldList updates. Each item in the form name array has this structure:

```
typedef struct
{
    char    name[PSF_FIELDNAMELEN + 1];
} PSFDEFNKEY;
```

Parameters

hSession The session handle assigned by PsfOpenSession when PeopleSoft connected to PSFORMS.DLL.

lpFormList

A pointer to a structure to hold the list of form names.
The form of the structure is:

```
typedef struct
{
    UINT    nNumForms; // The number of forms
    LPPSFDEFNKEY lpForms; // Form name array
} PSFFORMLIST;
```

Return Values

<i>Value</i>	<i>Code</i>	<i>Meaning</i>
PSF_OK	0	The function executed successfully.
PSF_NOMAIL	2	Mail system failure.
PSF_NODB	5	The forms database is not accessible.

PsfGetLastError

Syntax

```
int FAR PASCAL PsfGetLastError (HSESSION hSession, LPSTR lpszErrText,
    UINT uBufSize, UINT uErrCode);
```

Description

Gets the error message text associated with an error code.

PsfGetLastError gets the text of an error message based on the error code. The PeopleSoft system passes it the error code it received and a pointer to a text buffer. PsfGetLastError copies the corresponding error text into the buffer, up to the size specified by *uBufSize*.

Parameters

<i>hSession</i>	The session handle assigned by PsfOpenSession when PeopleSoft connected to PSFORMS.DLL.
<i>lpszErrText</i>	A pointer to a buffer where the function places the error text.
<i>uBufSize</i>	The size of the buffer that <i>lpszErrText</i> points to.
<i>uErrCode</i>	The error code to retrieve the error text for.

Return Values

<i>Value</i>	<i>Code</i>	<i>Meaning</i>
PSF_OK	0	The function executed successfully.
PSF_NOERR	9	No message was found for the specified error code.

PsfOpenSession

Syntax

```
int FAR PASCAL PsfOpenSession(LPSTR lpszUserName, LPSTR lpszPassword,
                              LPHSESSION lphSession);
```

Description

Connects the PeopleSoft system to the forms interface DLL using a specified login name and password.

PsfOpenSession logs in to the form software using the specified user name and password. It places a session handle in the buffer that *lphSession* points to. All subsequent API calls use this session handle.

PsfOpenSession always opens a new session, even when it could access a shared session.

Parameters

<i>lpszUserName</i>	A null-terminated string containing the user name to login to the forms software with.
<i>lpszPassword</i>	A null-terminated string containing the password for the user identified by <i>lpszUserName</i> .
<i>lphSession</i>	A pointer to a buffer for the session handle that this function assigns.

Return Values

Value	Code	Meaning
PSF_OK	0	The session was successfully opened.
PSF_NOSESS	4	The function was unable to establish a session.
PSF_NOSPEC	6	The forms database or server was not specified.

PsfSendForm

Syntax

```
int FAR PASCAL PsfSendForm (HSESSION hSession, LPPSFDEFN lpFormData);
```

Description

Sends a form through the mail system with the specified field values.

PsfSendForm sends a form via the specified open session. The *lpFormData* structure has values in all fields, including the data values in the field array it points to.

Parameters

hSession

The session handle assigned by PsfOpenSession when PeopleSoft connected to PSFORMS.DLL.

lpFormData

A pointer to a structure that contains form information. The form of the structure is:

```
typedef struct
{
    PSFDEFNKEY    formName;    // Name of the form
    WORD          wNumFields;  // Number of fields
    char          cSep;        // Separator
character
    LPPSFIELDVAL lpFields;    // Pointer to array
} PSFFORMDEFN;
```

Each item in the field array has this structure:

```
typedef struct
{
    char    szFldName[PSF_FORMNAMELEN + 1]; // Field name
    WORD    wFldSize;                        // Field size
    LPSTR    lpzStrVal;                      // Value
} PSFFIELDVAL;
```

Return Values

Value	Code	Meaning
PSF_OK	0	The function executed successfully.
PSF_NOFIELD	1	A field is missing or the wrong size.
PSF_NOMAIL	2	Mail system failure.
PSF_NOFORM	3	The form is not accessible.
PSF_NODB	5	The forms database is not accessible.

Index

A

- Action codes 7-3, 7-6
- API
 - repository 3-1
- APIs
 - for outgoing forms 12-3
- Application Attributes page 2-13. *See also* merchant integration, custom attributes
- Application load process 7-2
- application messaging
 - data synchronization 1-1
 - system-to-system workflow 1-1
 - when to use 1-1
- Application programming interfaces *See* APIs
- Auditing EDI processing 9-11

B

- business interlink
 - HTTPEnable.dll 2-14
 - in merchant integration 2-10, 2-14, 2-20, 2-23
 - when to use 1-3
- Business units
 - in external companies 7-19
 - setting up as trading partners 7-17

C

- Calculation options
 - EDI Agent 8-11, 8-19
- CD-ROM
 - ordering iii
- Codes
 - converting from EDI to PeopleSoft 7-3
- command line parameters
 - for PSTOOLS.EXE 6-1
 - format of 6-1
- component interface
 - when to use 1-2
- Conversion data profiles
 - assigning to trading partners 7-19
 - defined 7-6
 - defining 7-8
- Converting data
 - during EDI processing 7-3
- CreateSessionInformation *See* merchant integration, PeopleCode

- Customers
 - setting up as trading partners 7-18

D

- Data conversion
 - during EDI processing 7-6
 - EDI calculation options 8-11, 8-19
- database agents
 - when to use 1-8
- Database agents
 - adding to Process Scheduler 11-7
 - adding to workflow 11-4
 - assigning to panel groups 11-12
 - case study 11-13
 - command line 11-6
 - creating batches of online processes with 11-3
 - login parameters 11-7
 - monitoring database with 11-2
 - naming conventions for queries 11-5
 - specifying query to run 11-19
 - starting 11-12
 - starting multiple Message Agents 11-7
- Database Agents
 - defined 11-1
- DeleteSessionData *See* merchant integration, PeopleCode

E

- ECOUTMAP.SQC 9-3
- EDI
 - architecture 7-1
 - converting codes to PeopleSoft IDs 7-3
 - creating map definitions for 8-1
 - defining valid transactions 7-10
 - packages 9-12
 - reviewing and correcting errors 9-11
 - setting up trading partners 7-15
 - standard formats 7-2
 - transaction groups 9-12
- EDI Agents
 - managing 9-1
 - specifying how often to run 9-8, 9-11
 - starting 9-4
- EDI COORDINATOR role 9-4
- EDI manager
 - when to use 1-6
- EDI Manager 7-1, 8-1, 9-1
- EDIFACT format 7-2

Electronic commerce maps *See* Map definitions
 Electronic Data Interchange *See* EDI
 Entity codes 7-16
 Errors
 reviewing and correcting for EDI 9-11
 Event codes 7-4
 Event codes 7-3
 External business entities 7-19
 External trading partners *See also* Trading partners
 business units within 7-19
 setting up 7-18

F

File attachments
 from Lotus Notes 11-25
 File formats
 PeopleSoft Business Documents 8-2
 file layout 4-1
 creating 4-2
 CSV format 4-26
 CSV format considerations 4-27
 customizing 4-7
 date, time and datetime considerations 4-6
 example 4-16
 file field properties 4-12
 file record properties 4-9
 fixed format 4-25
 fixed format considerations 4-26
 naming 4-5
 properties 4-7
 segments vs. records 4-16
 XML format 4-27
 XML format considerations 4-28
 file layout objects and definitions
 when to use 1-5
 Forms
 reply methods for 11-19
 routings 12-1

G

GetAuthenticationParms *See* merchant integration,
 PeopleCode
 GetSessionData *See* merchant integration,
 PeopleCode

H

HTTPEnable.dll 2-14

I

Inbound transactions
 creating map definitions for 8-4

InsertSessionData *See* merchant integration,
 PeopleCode
 integration
 with external services 2-1. *See also* merchant
 integration
 integration technologies, overview 1-1
 Internal trading partners *See also* Trading partners
 setting up 7-17

L

Lotus Notes
 file attachments on forms 11-25
 Verify Originator ID option 11-19

M

Map definitions
 creating for EDI 8-1
 creating profiles for trading partners 8-20
 for inbound transactions 8-4
 for outbound transactions 8-14
 preparing for outbound EDI Agent 9-2
 Map profiles 8-20
 Map When 11-24
 Merchant Authentication page 2-9. *See also*
 merchant integration, connection and authentication
 Merchant BI Overrides page 2-10. *See also*
 merchant integration, business interlink
 Merchant Categories component 2-5. *See also*
 merchant integration, merchant categories
 Merchant Category page 2-12. *See also* merchant
 integration, merchant categories; merchant
 integration, merchant URLs
 merchant integration 2-1
 branded functionality 2-3
 business interlink 2-10, 2-14, 2-20, 2-23
 connection and authentication 2-9, 2-22, 2-32
 creating an MIP 2-20
 custom attributes 2-13, 2-24, 2-29
 development 2-14, 2-24
 hidden functionality 2-2
 HTML content 2-3
 implementing an MIP 2-30
 installation 2-3
 merchant account 2-19, 2-31
 merchant categories 2-5, 2-12, 2-20, 2-24, 2-33
 merchant profile 2-5, 2-7, 2-21, 2-31, 2-32
 merchant URLs 2-24
 MIP 1-6, 2-1, 2-3, 2-14
 PeopleCode 2-17, 2-25, 2-26, 2-28, 2-29
 process flow 2-1
 records and fields 2-15
 Secure Sockets Layer 2-5
 session information 2-5, 2-29
 Single Sign-On Framework 2-4, 2-25

- support 2-4
- user access 2-34
- user experience 2-2
- when to use 1-7
- merchant profile 2-5, 2-7. *See also* merchant integration. *See also* merchant integration
 - Application Attributes page 2-13, 2-24, 2-34
 - Merchant Authentication page 2-9, 2-22, 2-32
 - Merchant BI Overrides page 2-10, 2-23, 2-33
 - merchant categories 2-6, 2-20
 - Merchant Category page 2-12, 2-24, 2-33
 - Merchant Profile page 2-7, 2-21, 2-32
- Merchant Profile page 2-7. *See also* merchant integration, merchant profile
- message agent
 - when to use 1-7
- Message Agent 11-1
 - API setup 10-5
 - APIs 10-36
 - basic program setup 10-5
 - debugging 10-31
 - debugging -installation 10-31
 - debugging-administration 10-32
 - debugging-connecting 10-33
 - debugging-declaring functions 10-32
 - debugging-ProcessMessage 10-33
 - debugging-StartMessage 10-33
 - defined 11-15
 - deleting rows in scrolls 11-21
 - edit table processing 10-8
 - Examples 10-9
 - field mapping 10-4
 - handling file attachments 11-25
 - managing scroll levels using 10-2
 - mapping data into scrolls 11-20
 - multiple rows - adding/updating 10-3
 - multiple rows - retrieving multiple rows 10-3
 - multiple scroll levels - limitations 10-2
 - output data 11-21
 - processing 10-1
 - programming 10-4
 - replying to users 11-19
 - search dialog processing 10-6
 - sources of data 11-16
 - starting multiple instances 11-7
 - topic names for 11-7
 - Tricks of the Trade 10-35
 - triggering events through 11-2
 - troubleshooting 10-31
- Message definitions 11-1
 - creating valid field mappings 11-24
 - defined 11-16
 - icons for 11-16
 - order of fields in mappings 11-22
 - specifying panels for 11-18
 - specifying search records 11-19
 - types of 11-16

MIP *See* merchant integration. *See* merchant integration

O

- Object Linking and Embedding (OLE) *See* OLE automation
- OLE automation
 - Message Agent as server 10-37
- Open Query
 - API 5-3
 - ODBC driver 5-3
 - supported API calls 5-4
- Open Query ODBC Driver
 - architecture 5-2
- Outbound transactions
 - creating map definitions for 8-14
 - directory for files 7-13
 - preparing maps to run 9-2
- Outgoing Forms API
 - forms routing 12-1

P

- Packages (EDI)
 - reviewing auditing information 9-12
- Partner profiles 7-12
 - assigning to trading partners 7-19
- PeopleBooks
 - CD-ROM, ordering iii
 - printed, ordering iii
- PeopleCode
 - in merchant integration 2-17, 2-25
- PeopleSoft Business Document format 8-2
- Primary event codes *See* Event codes
- Process Scheduler
 - scheduling EDI Agents 9-4
- PSAUTHPARMS *See* merchant integration, records and fields
- PSFORMS.DLL 12-2
- PSMERCHANTAPP *See* merchant integration, records and fields
- PSMERCHANTCAT *See* merchant integration, records and fields
- PSMERCHBI *See* merchant integration, records and fields
- PSMERCHBIPARMS *See* merchant integration, records and fields
- PSSSESSIONDATA *See* merchant integration, records and fields
- PSTOOLS.EXE
 - command line parameters for 6-1
 - command line syntax 6-1
- Purpose codes *See* Event codes

Q

Queries

- for database agents 11-4
- specifying for database agents 11-19
- Queue control numbers 8-9

R

Records

- in PeopleSoft Business Documents 8-2

Registry

- adding sections for workflow 12-2

Reply methods

- for forms 11-19

repository

- bindings collection methods 3-8
- bindings collection properties 3-7
- bindings methods 3-9
- bindings properties 3-8
- ClassInfo collection methods 3-12
- ClassInfo collection properties 3-11
- ClassInfo properties 3-13
- discovery 3-1
- MethodInfo collection methods 3-13
- MethodInfo collection properties 3-14
- MethodInfo properties 3-15
- namespaces collection methods 3-9
- namespaces collection properties 3-9
- namespaces methods 3-11
- namespaces properties 3-10
- PeopleCode example 3-2
- properties 3-7
- PropertyInfo collection methods 3-16
- PropertyInfo collection properties 3-17
- PropertyInfo properties 3-17
- summary methods and properties 3-21
- using 3-1
- Visual Basic example 3-18

Routings

- defined 12-1

- Run controls 9-2

S

Scrolls

- managing with Message Agent 10-2

Search records

- specifying for message definitions 11-19
- Secondary event codes *See* Event codes
- Single Sign-On Framework *See* merchant integration, Single Sign-On Framework
- starting
 - PeopleSoft applications 6-1
- Starting
 - EDI Agents 9-4
 - Starting PSTOOLS.EXE 6-1

T

Third-party applications

- communicating with 12-2

- Trading Partner Conversion ID 7-7

Trading partners

- conversion profiles for 7-6
- creating map profiles for 8-20
- external 7-18
- internal 7-17
- partner profiles for 7-12
- setting up 7-15
- specifying valid transactions for 7-14

Transaction Groups (EDI)

- reviewing auditing information 9-12

Transactions

- available for trading partners 7-14
- defining for EDI 7-10
- processing new 8-1
- processing single 9-5
- reviewing auditing information for 9-11

V

- Value-added Network (VAN) 7-2

Vendors

- setting up as trading partners 7-18

W

Work records

- for EDI map definitions 8-5

X

- X.12 format 7-2