



PeopleTools 8.12 PeopleSoft
Business Interlink Design-Time
Plug-in Programming Guide
PeopleBook

PeopleTools 8.12 PeopleSoft Business Interlink Design-Time Plug-in Programming Guide
PeopleBookPeopleTools 8.12 PeopleSoft Business Interlink Design-Time Plug-in
Programming Guide PeopleBook

SKU MTBDr8SP1B 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	vii
Related Documentation	vii
Documentation on the Internet	viii
Documentation on CD-ROM	viii
Hardcopy Documentation	viii
Typographical Conventions and Visual Cues	ix
Comments and Suggestions	x

Chapter 1

Introduction to Business Interlinks

What is the Business Interlink Architecture	1-1
Running a Business Interlink	1-1
Creating a Business Interlink	1-4
Example 1: Calculating UPS Shipping Costs	1-4
Example 2: Creating a Calculate XML Design-Time Plug-in	1-7
List of Common Business Interlink Terms	1-9

Chapter 2

Designing Business Interlink Transactions and Classes

Listing Configuration Parameters	2-1
Example Design for Configuration Parameters	2-2
Listing Your Business Classes	2-2
Design for Freight Carrier Classes	2-2
Listing Your Business Transactions	2-4
Design For Freight Carrier Transactions	2-4
Diagram of the Calculate Cost Transaction	2-5

Chapter 3

Writing an XML Design-Time Plug-In

Using The XML Template File To Create Your XML Design-Time Plug-In	3-1
Example of an XML design-time plug-in for a Freight Carrier Business Interlink	3-2
Diagram of the Transactions	3-8

Writing The Tags in an XML Design-Time Plug-In	3-8
<general_info> Write the General Information for the Plug-in	3-9
<description> Write The Plug-in Description	3-9
<version> Write the Version Number	3-9
<image>	3-10
<driver_settings> List The Supported Business Interlink Types	3-10
<option> Write A Business Interlink Type	3-11
<relational_op> List The Query/Update Relational Operators	3-12
<logical_op> List The Query/Update Logical Operators	3-12
<operator>	3-13
<config_parameters> List The Configuration Parameters	3-13
<parameter> Write A Configuration Parameter	3-14
<BiDocValidate> Error Check BI Documents	3-14
<URL> Specify Business Interlink Runtime Plug-in File Location	3-15
<class_catalog> Write the Class Catalog	3-16
<category> Write The Categories For The Classes	3-17
<class> Write A Class	3-18
<member> Write Members For A Class	3-19
<trans_catalog> Write the Transaction Catalog	3-19
<category> Write The Categories For The Transactions	3-20
<transaction> Write A Transaction	3-20
List the Input Parameters <input_list>	3-21
Write an Input Parameter for a Transaction <input>	3-21
List the Output Parameters <output_list>	3-22
Write an Output Parameter for a Transaction <output>	3-22
Data Types	3-22

Chapter 4

Writing a XML Design-Time Plug-In using the pshttopenable Runtime Plug-In

Writing the Configuration Parameters for the pshttopenable runtime plug-in	4-1
<URL> Specify the pshttopenable Runtime Plug-in	4-1
Method: Specify the Method Parameters	4-2
SSL: Specify Secure Link	4-2
Accept_Type: Specify Secure Link	4-2
Content_Type: Specify Content Type Headers	4-3
USERID: Specify User ID	4-3
PASSWORD: Specify User Password	4-3
MerchantURL: Specify The Merchant URL	4-4
InDataType: Specify Input Data Type	4-4
OutDataType: Specify Output Data Type	4-4

XML_Validation: Check for Errors in XML Syntax	4-5
Input_File_Name: Debug Input Data	4-5
Request_File_Name: Debug Request Data	4-5
Simulated_Response_File_Name: Debug Request Data	4-6
Response_File_Name: Debug Request Data.....	4-6
Metatag: Debug Request Data.....	4-6
Writing the pshttpenable Runtime Plug-in Input Parameters	4-7
Writing the pshttpenable Runtime Plug-in Output Parameters.....	4-7
Writing the pshttpenable Runtime Plug-in Classes	4-7
Safe Characters	4-7
Example of XML Design-Time Plug-ins using pshttpenable.....	4-8
Diagram of the Example Transaction.....	4-8
pshttpenable Runtime Plug-in: Reading an XML String	4-14
Example of Design-Time Plug-in for Reading XML String.....	4-15
Example of PeopleCode for Reading XML String	4-20
XML Template for pshttpenable Runtime Plug-in	4-22

Chapter 5

Deploying your XML Design-Time Plug-In

Index

ABOUT THIS PEOPLEBOOK

This PeopleBook covers the concepts of Business Interlinks. It discusses how developers design the shape for a Business Interlink, and how they create Business Interlink XML Design-Time Plug-ins from that shape.

The Business Interlink XML Design-Time Plug-in is written in the XML markup programming language. An in-depth knowledge of XML is not needed, but some knowledge of XML syntax could be helpful. This PeopleBook shows and explains the syntax that you need to write a Business Interlink XML Design-Time Plug-in.

The chapters in this document are listed below.

Introduction to Business Interlinks provides an overview of Business Interlink Runtime Plug-ins.

Designing Business Interlink Transactions and Classes provides instructions for writing the Business Interlink transaction names and parameters and/or class names and data members that you need before you can write the design-time and run-time plug-ins.

Writing an XML Design-Time Plug-In provides instructions for writing a design-time plug-in using the XML language. It includes an example XML listing, and a description of each XML tag needed to write an XML design-time plug-in.

Writing a XML Design-Time Plug-In using the pshttpenable Runtime Plug-In provides instruction on writing a design-time plug-in that uses the Generic Plug-in. Some external systems receive data via HTTP in the form of an XML or HTML request, and then return data in the form of an XML or HTML response. The Generic Plug-in is a self-describing program that encapsulates such external systems.

Deploying your XML Design-Time Plug-In tells how to deploy design-time plug-ins for others to use.

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

Introduction to Business Interlinks

Business Interlinks enable you to perform component-based, realtime integration from PeopleSoft to external systems. Business Interlinks do this by creating synchronous transactions that allow PeopleSoft applications to pass data to and receive data from the external system in real time. You can use Business Interlinks to integrate PeopleSoft with external systems, with another PeopleSoft application, and systems on the internet web.

A transaction consists of a named command with optional named and typed inputs and outputs. The associated external system or the Business Interlink Plug-in understands this command.

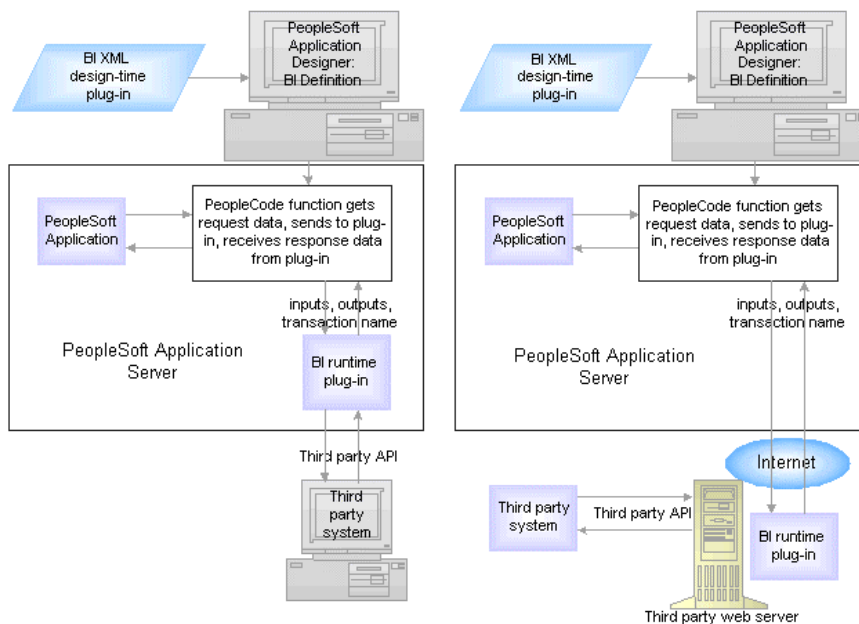
What is the Business Interlink Architecture

The Business Interlink Architecture provides a plug-in framework for PeopleSoft applications to invoke third party APIs over the Internet. Different vendors support different methods for exposing their APIs, including object technologies such as COM, COBRA, EJB; programming language specific interfaces for C or C++; or interfaces based on HTTP and XML. The Business Interlink framework provides a consistent framework for application developers to invoke external applications across this wide variety of technologies.

When a business event triggers the execution of a Business Interlink, the Component Processor synchronously calls the Business Interlink Processor, which in turn invokes the appropriate Business Interlink plug-in. The plug-in provides a wrapper around the third party API and is designed to support any type of interface binding (COM, CORBA, EJB, XML) exposed by the third-party interface. The third-party software could be hosted on the same machine as the PeopleSoft Internet Application Server, or on a separate machine on the other side of the world, invoked over the Internet.

Running a Business Interlink

The following diagram shows the typical Business Interlink architecture, using an XML design-time plug-in and a runtime plug-in.



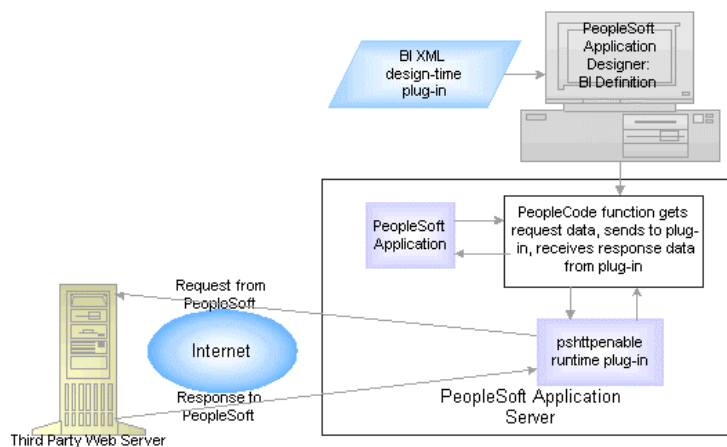
Business Interlink Architecture

When the Business Interlink Object is executed, the following actions take place:

1. A PeopleSoft Application triggers a PeopleCode event. For example, a user might, in a PeopleSoft page labeled “Shipping Time and Cost,” enter input values needed to calculate shipping time, and then press a button labeled “Calculate.”
2. The triggered event (pressing the “Calculate” button) passes the input values to a PeopleCode program, and then executes the PeopleCode program.
3. The PeopleCode program creates an Interlink Object, which contains the transaction name and inputs, and passes the Interlink Object to the Business Interlink runtime plug-in. The runtime plug-in can be located on:
 - A web server
 - The same PeopleSoft Application Server as the PeopleSoft Application.
4. The Business Interlink runtime plug-in provides the implementation of the transactions and/or data operations. It calls the external software system through its API, passing the input values from the Business Interlink Object. This external system can be located on:
 - An external server.
 - A web server.
 - The same PeopleSoft Application Server as the PeopleSoft Application.
5. The third party system performs operations based on those input values, and returns output values through its API to the runtime plug-in. These operations could be, for example, executing functions in the external system, or performing operations on a database in the external system.

- The Business Interlink runtime plug-in assigns output values to the Business Interlink Object (if there are outputs). If there are outputs, the PeopleCode program can assign the output values to PeopleSoft variables. For example, in a PeopleSoft page labeled “Shipping Time and Cost,” the output values could then be displayed upon that page.

There is a specialized type of Business Interlink design-time plug-in, which uses the pshttppenable runtime plug-in. Some external systems receive data via HTTP in the form of an XML or HTML request, and then return data in the form of an XML or HTML response. When your external system uses data in this manner, you can use the pshttppenable runtime plug-in. This means that a Business Interlink runtime plug-in need not be created for your external system. The following diagram shows the architecture for Business Interlinks using the pshttppenable runtime plug-in.



Business Interlink Architecture

When the Business Interlink Object is executed for a Business Interlink using a pshttppenable runtime plug-in, the following actions take place:

- A PeopleSoft Application with input values triggers a PeopleCode event.
- The triggered event passes the input values to a PeopleCode program, and then executes the PeopleCode program.
- The PeopleCode program creates an Interlink Object, which contains the input data and the transaction name, and passes the Interlink Object to the Business Interlink runtime plug-in. The runtime plug-in is the pshttppenable plug-in, located on the PeopleSoft application server.
- The pshttppenable runtime plug-in sends a request over the Internet to the third party web server, passing it input values from the Business Interlink Object.
- The third party web server performs operations based on those inputs, and returns a response over the Internet, passing back output values to the pshttppenable runtime plug-in.
- The Business Interlink runtime plug-in assigns output values to the Business Interlink Object (if there are outputs). If there are outputs, their values can be fetched and assigned to PeopleSoft variables.

Creating a Business Interlink

To allow users to create and run Business Interlinks, developers must perform the following tasks. This manual describes the tasks that are in **bold**.

1. A developer designs the shape of a transaction(s) (inputs, outputs, name). For example, the action could be telling UPS to calculate the cost of shipping.
2. A developer writes a Business Interlink design-time plug-in that implements the shape of the transaction(s). This plug-in is written in the XML markup language.
3. A developer writes a Business Interlink runtime plug-in to implement the transaction: passing the inputs to an external system and receiving the outputs from the external system.
4. The design-time plug-in and the runtime plug-in are deployed for use by PeopleSoft applications.
5. A PeopleSoft Application Developer creates a Business Interlink Definition, which creates a specific shape for a particular PeopleSoft application. For example, the application might be created, or modified, to be able to connect to UPS and calculate shipping costs.
6. A PeopleSoft Application Developer writes a PeopleCode program to call the Business Interlink object that is created for the Business Interlink definition.
7. A user can now use the PeopleSoft application to run the Business Interlink. For example, the user can now connect to UPS and calculate shipping costs.

Example 1: Calculating UPS Shipping Costs

The Shipping Time & Cost example shows how Business Interlinks can be used. Shipping Time & Cost use the Business Interlink plug-ins that were written for UPS.

http://dgiammon042400/iclientservlet.run - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://dgiammon042400/iclientservlet.run

Exit Help

Shipping Time & Cost
United Parcel Service

Ground Time In-Transit

*Origin:

*Destination:

QuickCost

*Country Origin: United States *Zip:

*Country Destination: United States *Zip:

*Drop off/Pickup Type: One Time Pickup *Weight:

*Packaging: UPS Express Bo lbs.

Service Type	Guaranteed By	Commercial Rate

Local intranet

Shipping Time & Cost Page

In the Shipping and Time page, you can trigger two different Business Interlinks that call out to the UPS shipping and tracking Web site to calculate:

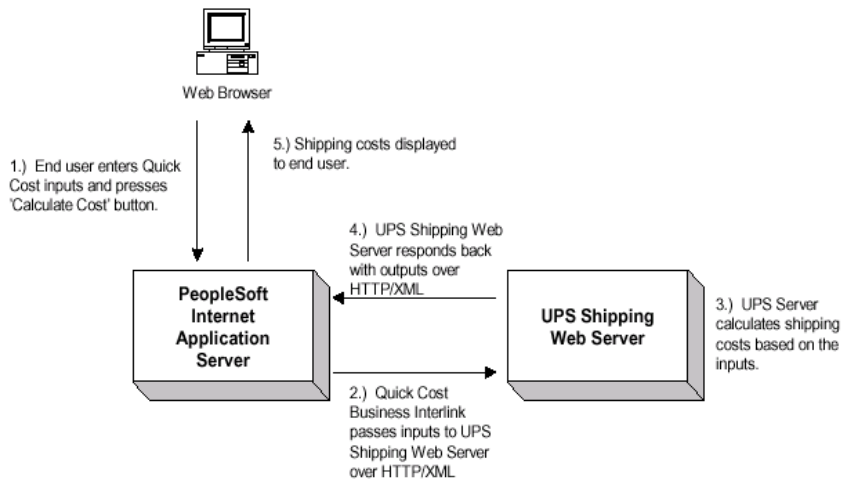
- Ground Time-in-Transit to determine how long it will take to deliver the package.
- Quick Cost to calculate how much it will cost to ship the package based on the shipping service type.

For Quick Cost, you first enter Origin Country and Postal Code, Destination Country, Postal Code, Packaging information, and then you press the Calculate button. At this point, the Business Interlink is invoked on the Application Server, and it issues an HTTP request out to the UPS Web site, which calculates the shipping costs and returns the information back to the PeopleSoft application. Here are the results of a Quick Cost calculation.

Service Type	Guaranteed By	Commercial Rate
UPS Next Day Air Early A.M.	8:30 A.M. - Next Day	\$ 70.25
UPS Next Day Air	10:30 A.M. - Next Day	\$ 45.25
UPS Next Day Air Saver	3:00 P.M. - Next Day	\$ 39.75
UPS 2nd Day Air A.M.	12:00 P.M. - 2 Days	\$ 27.30
UPS 2nd Day Air	End of Day - 2 Days	\$ 24.30
UPS 3 Day Select	End of Day - 3 Days	\$ 16.70
UPS Ground		\$ 7.24

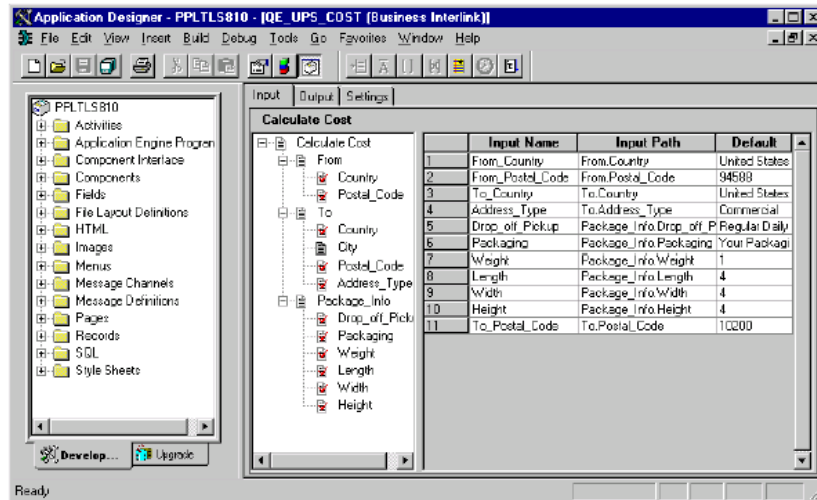
Shipping Time & Cost Page: Calculate Cost Business Interlink

This diagram explains the processing flow of this example in more detail.

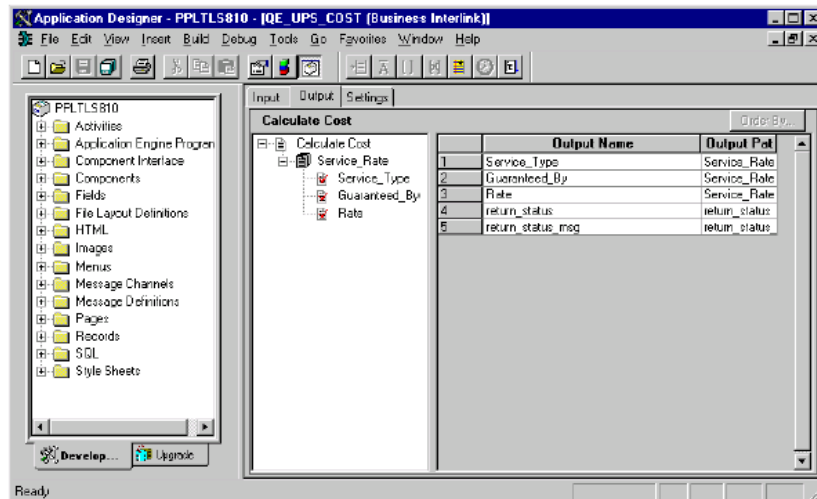


Calculate Cost Business Interlink Processing Flow

In the PeopleTools Application Designer, you can view the Business Interlink Definition for the Quick Cost application. A Business Interlink consists of a set of inputs and outputs. The first screen shows the inputs, and the second shows the outputs.



Calculate Cost Business Interlink Inputs



Calculate Cost Business Interlink Outputs

Example 2: Creating a Calculate XML Design-Time Plug-in

Following is a simplified XML design-time plug-in example for a transaction named Calculate Cost, with inputs of From, To, and Package_Info, and outputs of Service_Rate.

```
<?xml version="1.0" ?>

<interface_driver>

  <general_info>

    <description>UPS services</description>
```

```
<version>1</version>

<comments>UPS plug-in</comments>

<image>ups.bmp</image>
</general_info>

<config_parameters>
  <URL>file://ups.dll</URL>
</config_parameters>

<trans_catalog>
  <category name="UPS transactions">
    <transaction name="Calculate Cost">
      <input_list>
        <input name="From"
          type="string"
        <input name="To"
          type="string"
        <input name="Package_Info"
          type="string"
      </input_list>
      <output_list>
        <output name="Service_Rate"
          type="string"
      </output_list>
    </transaction>
  </category>
</trans_catalog>
</interface_driver>
```

List of Common Business Interlink Terms

Application Designer: The integrated development environment used to develop PeopleSoft applications.

Basic Type: A data type such as an integer or string.

Business Interlink Definition: A definition encapsulating an external Transaction or Query and providing a set of generically typed input/outputs that can be assigned to PeopleCode variable or Record Fields at runtime. A Business Interlink Definition is added to the Application Designer's objects at the same level as Fields, Records, Pages, etc.

XML Design-Time Plug-in: An XML file that, when coded for an external system, encapsulate that external system and provide a catalog of Transactions, Classes and Criteria specific and meaningful to that external system. This functionality can also be written using a set of C++, Visual Basic, or other high-level language methods.

Business Interlink Framework: The framework for integrating any external system with PeopleTools application objects. It is composed of the following components: 1) An External System, 2) Generic definitions for a Transaction/Query command interfaces, 4) Business Interlink Definitions, 4) Business Interlink Plug-in.

Business Interlink Object: an instantiation based on a Business Interlink Definition. Actual data can be added to the inputs of the Business Interlink Objects once the appropriate bindings are provided. The Business Interlink Object can be executed to perform the external service. Once a Business Interlink Object is executed, the user of that object can retrieve the outputs of the external service. The Business Interlink Objects use buffers to receive input and send output. When a Business Interlink Object is executed, the transaction/query/class associated to the Business Interlink Object will be executed once per each row of the input buffers corresponding to the input Records. If there is only one row, after appropriate substitution by the driver, it is executed only once.

Runtime Plug-in: A set of C++, Visual Basic, or other high-level language methods that, when coded for an external system, encapsulate that external system and provide the execution methods to match the Business Interlink Design-Time Plug-in.

Catalog: the list of transactions, classes, and queries used to interface to the external system. Integration users are presented with this list when they pick the type of Business Interlink Plug-in they are going to use. There are four types of catalogs:

- Transaction catalog: lists transactions used to interface to the external system. See Transaction.
- Class catalog: lists classes used to interface to an external system. A class contains data members of basic types and/or objects that are typed after another class. A Class can also contain lists of basic types or objects.
- Operator catalog: lists query operators used to query the external system. These query operators are used to query the classes in the class catalog.
- Configuration parameter catalog: Used to configure an external system with PeopleSoft. For example, it might set up configuration and communication parameters for an external server.

External System: Any system that is not directly compiled with the PeopleTools servers.

PeopleCode: PeopleSoft's proprietary language; it is executed by the PeopleSoft Application Processor. PeopleCode generates results based upon specific actions, based upon existing data or the actions of a user. Business Interlink Objects are executed by calling the execute() method from PeopleCode. This makes external services available to all PeopleSoft applications wherever PeopleCode can be executed.

PeopleCode Event: An action that an end-user takes upon an object, usually a Record Field, that is referenced within a PeopleSoft page.

Query: a set of data members that are selected from a Class catalog (provided by the Business Interlink Plug-in) as well as a generic form of Criteria. The criteria are composed of <left-hand-side> <Relational Operator> <right-hand-side> statements that can be concatenated using a set of logical operators. All operators and class catalogs are dynamically provided through the Business Interlink Plug-in.

Transaction: a named command with optional named and typed inputs and outputs. The associated external system or the Business Interlink Plug-in understands this command. The types of inputs and outputs are based on a set of generic types.

Shape: For a transaction, the set of inputs and outputs for that transaction. For a class, the data members of that class.

CHAPTER 2

Designing Business Interlink Transactions and Classes

When you use Business Interlinks with your system, you must design a set of Business Interlink Transactions and/or Classes. A **Business Transaction**, or transaction, is a named command with optional named and typed inputs and outputs. A **Business Class**, or class, is a definition of a data structure. If you map the data structures in your external system to Business Classes, you can use the Business Interlink Framework to perform queries, adds, deletes, and updates on your data.

You should be familiar with the PeopleSoft components, pages, or batch programs that will be calling your Business Interlink Plug-in, and the input and output data they will need, in order to determine all the Business Transactions and Business Classes needed to be supplied by the plug-in.

You will determine the following:

- If you need transactions, classes, or both.
- The list of transaction names, and the inputs and outputs for the transactions.
- The list of class names, and the data members for the classes.
- If you need a list of Configuration Parameters: global parameters that you can easily access within the execution methods of your Business Interlink Plug-in.

Listing Configuration Parameters

You might need to supply Configuration Parameters. Configuration parameters are used as global parameters that you can easily access within the execution methods of your Business Interlink Plug-in. For example, an email system would often need information allowing access to an email server, so its configuration parameters could be a mail server type (STMP or Pop3), a logon name, a password, and an email address.

There is one configuration parameter that is standard: the URL configuration parameter specifies where your Business Interlink runtime plug-in is located.



For more information about the URL configuration parameter, see <URL> Specify Business Interlink Runtime Plug-in File Location.

If you use configuration parameters other than the URL configuration parameter, determine the following information for each configuration parameter:

- The name.
- The data type.

Example Design for Configuration Parameters

The Freight Carrier example only uses the URL configuration parameter.

Listing Your Business Classes

You can use classes when you want to organize the input and output parameters for your transactions. For example, in a transaction for Freight Carrier that calculates the cost of sending a package, you may use classes to organize the package information: its origin, destination, type, and account information.

You can also use classes to map to data in your external system so that you can perform one or more of the following methods on that data: query, add, update, and delete. For example, if you have a database with Customer records, you may want to be able to use all four methods: query for customer information, update a customer's information, add a customer to the database, and delete a customer from the database. Or, if your external system provides access to its application objects through a run-time object engine, or a relational storage, you want to define a set of classes, each with their data members, some of which can point to other classes.

If you use Business Classes, determine the following information for each class:

- The class name.
- The names and types of the data members for that class.

Design for Freight Carrier Classes

The Freight Carrier classes are used to help organize the input and output parameters for the Freight Carrier transactions.

Origin, the class containing information about the origin location of a Freight Carrier package, can be designed as follows.

<i>Data Member Name</i>	<i>Data Type</i>
Country	Enumeration. The enumerated types are: United States, Puerto Rico
Postal_Code	String

Destination, the class containing information about the destination location of a Freight Carrier package, can be designed as follows.

Data Member Name	Data Type
Country	Enumeration. The enumerated types are: Argentina,Australia,Aruba,Brazil,Bosnia,Canada,China,Costa Rica,Finland,France,Germany,Greece,Hong Kong,Iran,Italy,Israel,Japan,Korea,Mexico,Russia,Spain,Taiwan,United Kingdom,United States,Zambia
City	String
Postal_Code	String
Address_Type	Enumeration. The enumerated types are: Commercial, Residential

Package_Information, the class containing information about a Freight Carrier package, can be designed as follows.

Data Member Name	Data Type
Drop_off_Pickup	Enumeration. The enumerated types are: Regular Daily Pickup,On Call Air,One Time Pickup,Letter Center,Customer Counter
Packaging	Enumeration. The enumerated types are: Your Packaging,UPS Letter Envelop,UPS Tube,UPS Express Box,UPS Worldwide 25KG Box,UPS Worldwide 10KG Box
Weight	Integer
Length	Integer
Width	Integer
Height	Integer

Service Rate, the class containing information about the Freight Carrier service rates, can be designed as follows.

Data Member Name	Data Type
Service_Type	String
Guarnteed_By	String
Rate	String

Listing Your Business Transactions

If the purpose of your external system is to perform functions (other than update, add, delete, query of structured data), you will likely use Business Transactions.

If your external system already provide a set of transactions with well-defined interfaces, you are almost done with this step. You will only need to organize a list of transaction names, each with a set of (optionally hierarchical) named and typed inputs and outputs. On the other hand, if there are no well-defined transactions for interfacing, you may need to define a set of logical transactions that wrap around the lower level functions you are trying to expose.

If you will use Business Transactions, decide on the following information for each transaction:

- The transaction name
- What the transaction does
- What input and output parameters the transaction uses, and their data types

Design For Freight Carrier Transactions

Calculate Cost, the transaction that calculates the cost of sending a Freight Carrier package, can be designed as follows.

<i>Input Parameter Name</i>	<i>Data Type</i>
From	Origin class
To	Destination class
Package_Info	Package_Information class

<i>Output Parameter Name</i>	<i>Data Type</i>
Service_Rate	Service Rate class

Time-in-Transit, the transaction that calculates the time taken to ship an item, can be designed as follows.

<i>Input Parameter Name</i>	<i>Data Type</i>
From	Origin class
To	Destination class

<i>Output Parameter Name</i>	<i>Data Type</i>
From	Origin class

Output Parameter Name	Data Type
To	Destination class
Transit_Time	String

Tracking, the transaction that tracks a shipped item, can be designed as follows.

Input Parameter Name	Data Type
Tracking_Number	String

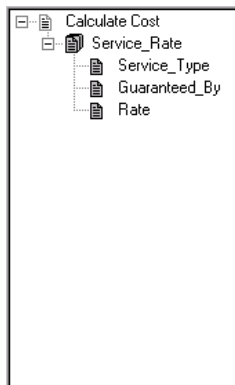
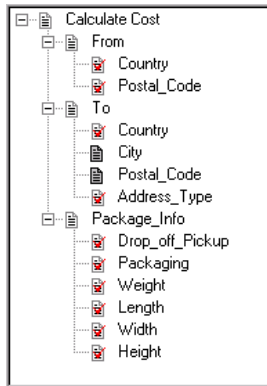
Output Parameter Name	Data Type
Tracking_Number	String
Status	String
Date	Date



For more information about writing the XML design-time plug-in to define these class and transaction catalogs, see Writing an XML Design-Time Plug-In.

Diagram of the Calculate Cost Transaction

Here is a diagram of the Calculate Cost(Domestic) transaction resulting from the designs in this chapter.



Design of Calculate Cost Transaction

CHAPTER 3

Writing an XML Design-Time Plug-In

Once you have designed your transactions and classes by naming the transactions/classes and their parameters, you introduce your transactions and classes to the Business Interlink framework. Before you can manifest your transactions as Business Interlink Definitions, you introduce them to the PeopleTools design-time environment. This is accomplished by supplying an XML design-time plug-in. This plug-in is used to define and save Business Interlink Definitions. It can also be used to test the Business Interlink Definition by simulating the execution of the corresponding Business Interlink Objects, if you supply default output values.

After you have decided what services that your system should support, check the services to see if their catalogs are static. A static catalog means that the transactions and classes are predefined; the input/output parameters for a transaction, or the data members for a class, are not changeable in data type or number of parameters/members.

When you use static catalogs, you can supply an XML design-time plug-in. The XML design-time plug-in uses tags for expressing all the information required for defining the interface to your transactions.



For comparison, a dynamic catalog would mean that the transactions and classes are changeable; a plug-in to a database would allow the members of a class to be changed, also a new class can be added or deleted. The PeopleSoft Business Interlink Runtime Plug-in Programming Guide contains a chapter, Writing the Design-Time Functionality Using Dynamic Catalog Methods, that describes how to write design time functionality when you do not write an XML design-time plug-in.

Using The XML Template File To Create Your XML Design-Time Plug-In

When you write your XML design-time plug-in, you can copy the file `template.xml`, and edit that copy to create your own XML design-time plug-in. The `template.xml` file contains most of the structure that you need for your XML design-time plug-in. Copy the `template.xml` file from the following directory, and then use a text editor to create your XML design-time plug-in.

The template is stored at the following location:

```
<PS_HOME>\SDK\src\C++\TEMPLATES
```

For examples of completed XML design-time plug-ins, search the following directories for file names ending with `.xml`:

```
<PS_HOME>\SDK\src\C++\Samples
```

Example of an XML design-time plug-in for a Freight Carrier Business Interlink

This section contains an XML design-time plug-in for a Freight Carrier plug-in. It contains four classes (Origin, Destination, Package_Information, Service_Rate) and three transactions (Calculate Cost, Tracking, Time-in-Transit). Note that the classes are used as inputs and outputs for the transactions.



For more information about the design of the transactions and classes used for this example, see Design for Freight Carrier Classes and Design For Freight Carrier Transactions.

```
<?xml version="1.0" ?>

<interface_driver>

    <general_info>

        <description>UPS services</description>

        <version>1</version>

        <comments>UPS plug-in</comments>

        <image>ups.bmp</image>

    </general_info>

    <driver_settings>

        <option type="static_catalog" supported="true"/>

        <option type="transaction" supported="true"/>

        <option type="input_class_expandable" supported="true"/>

    </driver_settings>

    <relational_op>

    </relational_op>

    <logical_op>
```

```

    </logical_op>
</driver_settings>

```

```

<config_parameters>
    <URL>file://ups.dll</URL>
</config_parameters>

```

```

<class_catalog>
    <category name="Internal Class">

```

```

        <class name="Origin">
            <member name="Country"
                type="enum(United States,Puerto Rico) "
                default="United States"
                required="true"/>
            <member name="Postal_Code"
                type="string"
                default="94588"
                required="true"/>
        </class>

```

```

        <class name="Destination">
            <member name="Country"
                type="enum(Argentina,Australia,Aruba,Brazil,Bosnia,Canada,China,Costa
                Rica,Finland,France,Germany,Greece,Hong
                Kong,Iran,Italy,Israel,Japan,Korea,Mexico,Russia,Spain,Taiwan,United
                Kingdom,United States,Zambia) "
                default="United States"
                required="true"/>

```

```

    <member name="City"

        type="string"

        default="New York"

        required="false"/>

    <member name="Postal_Code"

        type="string"

        default="10200"

        required="false"/>

    <member name="Address_Type"

        type="enum(Commercial,Residential) "

        default="Commercial"

        required="true"/>

</class>

<class name="Package_Information">

    <member name="Drop_off_Pickup" type="enum(Regular Daily Pickup,On
    Call Air,One Time Pickup,Letter Center,Customer Counter) "

        default="One_Time_Pickup" required="true"/>

    <member name="Packaging" type="enum(Your Packaging,UPS Letter
    Envelop,UPS Tube,UPS Express Box,UPS Worldwide 25KG Box,UPS Worldwide 10KG Box
    )"

        default="UPS_Express_Box" required="true"/>

    <member name="Weight"

        type="int"

        default="1"

        required="true"/>

    <member name="Length"

        type="int"

        default="4"

        required="true"/>

    <member name="Width"

```



```
        type="int"

        default="4"

        required="true"/>
<member name="Height"

        type="int"

        default="4"

        required="true"/>
</class>

<class name="Service Rate">

    <member name="Service_Type"

        type="string"

        default="UPS Next Day Air Early AM"/>

    <member name="Guaranteed_By"

        type="string"

        default="8:00 AM Next Day"/>

    <member name="Rate"

        type="string"

        default="50.00"/>

</class>

</category>

</class_catalog>

<trans_catalog>

    <category name="UPS transactions">

        <transaction name="Tracking">

            <input_list>

                <input name="Tracking_Number"
```

```
        type="string"
        required="true"/>
</input_list>
<output_list>
    <output name="Tracking_Number"
        type="string"
        default="1Z897X430397192061"/>
    <output name="Status"
        type="string"
        default="Delivered"/>
    <output name="Date"
        type="date"
        default="01/05/1998 16:12:00"/>
</output_list>
</transaction>
```

```
<transaction name="Time-in-Transit">
    <input_list>
        <input name="From"
            type="string"
            required="true"/>
        <input name="To"
            type="string"
            required="true"/>
    </input_list>
    <output_list>
        <output name="From"
            type="string"
            default="Pleasanton"/>
```

```
<output name="To"
      type="string"
      default="San Jose"/>
<output name="Transit_Time"
      type="string"
      default="1"/>
</output_list>
</transaction>

<transaction name="Calculate Cost">
  <input_list>
    <input name="From"
          type="object"
          classname="Origin"/>
    <input name="To"
          type="object"
          classname="Destination"/>
    <input name="Package_Info"
          type="object"
          classname="Package_Information"/>
  </input_list>
  <output_list>
    <output name="Service_Rate"
          type="list_object"
          classname="Service Rate"/>
  </output_list>
</transaction>

</category>
```

```

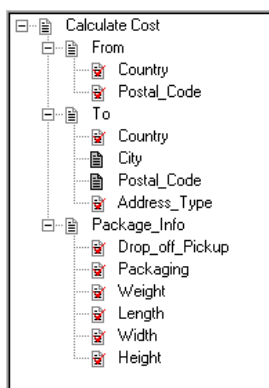
    </trans_catalog>

</interface_driver>

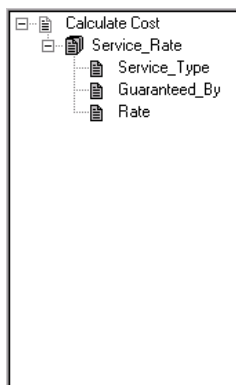
```

Diagram of the Transactions

Here is a diagram of the Calculate Cost transaction resulting from the XML code in this chapter.



Design of Calculate Cost Transaction Inputs



Design of Calculate Cost Transaction Outputs

Writing The Tags in an XML Design-Time Plug-In

This section of the chapter explains each tag that you write in the XML design-time plug-in, and provides an example for each tag.

The main tag for the Business Interlink XML design-time plug-in is **<interface_driver>**. It contains the **<general_info>**, **<driver_settings>**, **<config_parameters>**, **<class_catalog>**, and **<trans_catalog>** tags.

<general_info> Write the General Information for the Plug-in

In the **<general_info>** tag, write a description of your plug-in and the version number for your plug-in, and optionally the last time the XML design-time plug-in was updates and any comments for this XML design-time plug-in.

Here is an example of how the **<general_info>** tag is coded for a Freight Carrier plug-in. The description is “FEDEX services”, and the version number is 1.

```
<general_info>

    <description>UPS services</description>

    <version>1</version>

    <comments>UPS plug-in</comments>

    <image>ups.bmp</image>

</general_info>
```

<description> Write The Plug-in Description

In the **<description>** tag, write a short description of your XML design-time plug-in. The **<description>** tag must be placed within a **<general_info>** tag.

Here is an example of the **<description>** tag. The description is “UPS services”.

```
<description>UPS services</description>
```

Within the Application Designer, the description is displayed within the New Business Interlink page, and at the top of each Interlink page within the Application Designer.

<version> Write the Version Number

In the **<version>** tag, write the version number for your XML Design-Time Plug-In. The **<version>** tag must be placed within a **<general_info>** tag. The version number is used by the Business Interlink Framework to determine the version of your plug-in. If there is more than one version of your plug-in, this number is needed to identify the plug-in.

Here is an example of the **<version>** tag. The version number is 1.

```
<version>1</version>
```

The version number allows you to write future versions of your XML design-time plug-in, and to uniquely identify each version with a number. The version number for the XML design-time plug-in will be compared to the version number for the C++ Business Interlink runtime plug-in that you also write; there can be more than one set of C++ Business Interlink runtime plug-ins per XML design-time plug-in.



For more information about the version numbers for the C++ Business Interlink Plug-in Class, refer to Writing GetVersion for Your Business Interlink Plug-in Class and Writing IsVersionCompatible for Your Business Interlink Plug-in Class.

<image>

In the **<image>** tag, name a graphic file that you want to use with this Business Interlink. The graphic appears in the Business Interlink property box in the Application Designer.

This graphic file must be stored in the same location as where the XML design-time plug-in is deployed.

Here is an example of how the **<image>** tag could be coded.

```
<image>ups.bmp</image>
```

<driver_settings> List The Supported Business Interlink Types

In the **<driver_settings>** tag, list the Business Interlink types that your plug-in will support, and list any relational operators and logical operators if your plug-in supports them.

Here is an example of how the **<driver_settings>** tag could be coded. The supported Business Interlink type is transaction.

```
<driver_settings>

  <option type="static_catalog" supported="true"/>

  <option type="transaction" supported="true"/>

  <option type="input_class_expandable" supported="true"/>


  <relational_op>

  </relational_op>


  <logical_op>

  </logical_op>
```

```
</driver_settings>
```

<option> Write A Business Interlink Type

Write an **<option>** tag for each Business Interlink type that your plug-in supports. The **<option>** tag must be placed within a **<driver_settings>** tag.

Here are examples of **<option>** tags that support the Business Interlink types of transaction, static catalog (static catalog being required in the XML design-time plug-in), and input_class_expandable, so that inputs can be classes.

```
<option type="static_catalog" supported="true"/>
```

```
<option type="transaction" supported="true"/>
```

```
<option type="input_class_expandable" supported="true"/>
```

The valid values for *type* are:

type	Definition
static_catalog	When you write an XML design-time plug-in, you have static catalogs. You must include the following <option> tag with every XML design-time plug-in for a Business Interlink Plug-in: <option type="static_catalog" supported="true"/>
transaction	Your plug-in supports transaction Business Interlinks.
object_query	Your plug-in supports query Business Interlinks.
object_add	Your plug-in supports add Business Interlinks.
object_update	Your plug-in supports update Business Interlinks.
object_delete	Your plug-in supports delete Business Interlinks.
order_by_desc	If you use the object_query type, you can use order_by_desc for descending order. This type allows a query Business Interlink Object to list query results in descending order. The type order_by_desc is only used with query Business Interlink types.
order_by_asc	If you use the object_query type, you can use order_by_asc for ascending order. This type allows a query Business Interlink Object to list query results in ascending order. The type order_by_asc is only used with query Business Interlink types.
input_class_expandable	You must include the following <option> tag with every XML design-time plug-in that contains, within a <transaction> tag, an <input> tag that is of type object. The object type allows inputs to be classes by pointing to a <class> tag.

	<p>For more information about the <input> tag, see Write an Input Parameter for a Transaction <input>.</p> <hr/> <pre><option type="input_class_expandable" supported="true"/></pre>
output_class_expandable	<p>You must include the following <option> tag with every XML design-time plug-in that contains, within a <transaction> tag, an <output> tag that is of type object. The object type allows outputs to be classes by pointing to a <class> tag.</p> <hr/> <p>For more information about the <output> tag, see Write an Output Parameter for a Transaction <output>.</p> <hr/> <pre><option type="output_class_expandable" supported="true"/></pre>

<relational_op> List The Query/Update Relational Operators

If you use object_query or object_update types, then you can write relational operators by using the **<relational_op>** tags. Use the **<relational_op>** tag only if you have relational operators. The **<relational_op>** tag must be placed within a **<driver_settings>** tag. The **<relational_op>** tag is only valid with the Interlink types of object query or object update.

The following example shows how you would set relational operators for + and !=.

```
<relational_op>

  <operator name="+" />

  <operator name="!=" />

</relational_op>
```

<logical_op> List The Query/Update Logical Operators

If you use object_query or object_update types, then you can write logical operators by using the **<logical_op>** tags. In these tags, include the name of the operator. Use the **<logical_op>** tag only if you have logical operators. The **<logical_op>** tag must be placed within a **<driver_settings>** tag. The **<logical_op>** tag is only valid with the Interlink types of object query or object update.

The following example shows how you would set logical operators for OR and AND.

```
<logical_op>

  <operator name="OR" />
```



```

    <operator name="AND" />

</logical_op>

```

<operator>

Write an **<operator>** tag for each relational operator and logical operator that your plug-in supports. The **<operator>** tag must be placed within a **<relational_op>** tag or a **<logical_op>** tag.

The following example shows how you would set a relational operator for !=.

```

<operator name="!=" />

```

<config_parameters> List The Configuration Parameters

In the **<config_parameters>** tag, write the configuration parameters for your plug-in. Configuration parameters contain data that can be used in a variety of ways, depending upon your system. A common use is for the configuration parameters to be a set of global variables that you would use when you write the execution code for your plug-in. Another common use for configuration parameters is data that helps connect your external service to PeopleSoft.

Here is an example of how the **<config_parameters>** tag is coded for a Freight Carrier plug-in.

```

<config_parameters>

    <URL>file://ups.dll</URL>

</config_parameters>

```



Whenever you specify a UNIX directory name as a configuration parameter, you must end the directory name with a backslash /.

Here is an example of how the **<config_parameters>** tag could be coded for an email plug-in. This example shows how two configuration parameters are coded. The configuration parameter SMTP_MAIL_SERVER is of type string, has a default value of st-sun13.peoplesoft.com, and is a required configuration parameter. The configuration parameter LOGIN_NAME is of type string, has a default value of certora, and is a required configuration parameter.

```

<config_parameters>

    <parameter name="SMTP_MAIL_SERVER"

        type="string"

        default="st-sun13.peoplesoft.com"

```

```

        required="true"/>

<parameter name="LOGIN_NAME"

    type="string"

    default="certora"

    required="true"/>

</config_parameters>

<URL>file//:myplugin.dll</URL>

```

<parameter> Write A Configuration Parameter

Write a **<parameter>** tag for each configuration parameter that your plug-in uses. The **<parameter>** tag must be placed within a **<config_parameters>** tag. A **<parameter>** tag contains a name and type, and an optional default value.

Here is a **<parameter>** tag for a required configuration parameter that is a character string named LOGIN NAME.

```

<parameter name="LOGIN_NAME"

    type="string"

    default="certora"

    required="true"/>

```



For more information about the valid values you can use for *type*, see Data Types.

<BiDocValidate> Error Check BI Documents

When set to ON, the PeopleCode program that runs a BIDocs object is checked to see if it exists before adding/getting values from it. A BIDocs object is the most common method used by PepperCode to access the input and output data for Business Interlinks.



For more information about the PeopleCode program for Business Interlinks, see *PeopleSoft Business Interlink Application Developer Guide*, Running the Business Interlink Object.

For example, if the BIDoc object “Rate” does not exist, and BIDocValidate is set to ON, then the following line of PeopleCode will cause an error:

```
&ret = &biDocs.GetValue("Rate", &RATE);
```

The default value is ON.

<URL> Specify Business Interlink Runtime Plug-in File Location

Write a <URL> tag if you want to specify where the Business Interlink runtime plug-in is located. Here is an example of how the <URL> tag is coded for a Freight Carrier plug-in, telling that the runtime plug-in is a file named psfedex.dll.

```
<URL>file://ups.dll</URL>
```

If you supply a <URL> tag, and the plug-in is not found in the location specified by the <URL> tag, an error message will occur when a user attempts to use this plug-in. The plug-in can be specified as follows:

```
<URL>file://runtime_plug-in_name.dll</URL>
```

runtime_plug-in_name the name of the Business Interlink runtime plug-in.

This will find a runtime plug-in named *runtime_plug-in_name.dll* in the InterfaceDrivers directory. The InterfaceDrivers directory is located in the PeopleTools installation.

```
<URL>file://path/runtime_plug-in_name.dll</URL>
```

path the path to your Business Interlink runtime plug-in file.

runtime_plug-in_name the name of your runtime plug-in file.

This will find the runtime plug-in named *runtime_plug-in_name.dll* at the specified path on the computer where PeopleTools is installed.

```
<URL>http://webserver/servletname</URL>
```

webserver the URL where the Business Interlink installation is located. (Windows NT only.)

servletpath the path to a servlet on the Business Interlink installation.

servletname varies depending upon the Business Interlink installation for the web server. For Microsoft IIS, the default name is:

```
BusInterlink.asp
```

For Apache Web Server, the default name and path is:

```
Servlets\BusInterlinkServlet
```

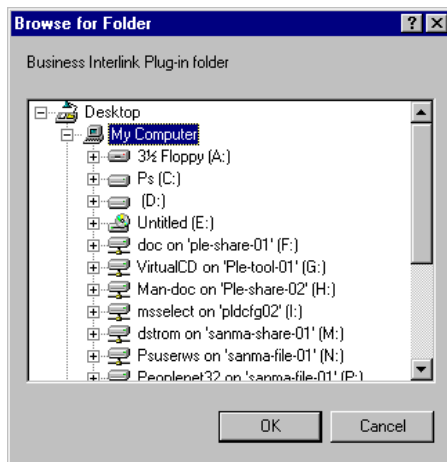
For *servletname* to work, the runtime plug-in must use the same name as the name of the XML design-time plug-in. For example, if runtime is *fcarrier.dll*, the design-time must be *fcarrier.xml*.



For more information about the runtime plug-in, see the PeopleSoft Business Interlink Runtime Plug-in Programming Guide.

The method that the Business Interlink Framework uses to find your runtime plug-in DLL file is:

1. It looks in the location specified by the **<URL>** tag.
2. If there is no **<URL>** tag, it looks in the location specified by the Browse For Folder page, which is accessed by pressing the Location button in the New Business Interlink page.



Browse For Folder Page

3. If the search location button is not used, it looks in the InterfaceDrivers directory:

```
PeopleTools_directory\ptdvl\bin\client\win86\InterfaceDrivers
```

where *PeopleTools_directory* is the directory where PeopleTools is installed.

<class_catalog> Write the Class Catalog

In the **<class_catalog>** tag, write the class catalog that you want to use with your plug-in. The class catalog provides the categories for the classes, the names of the classes, the names of the data members for each class, the data type of each data member, and optional default values for the data members.

```
<class_catalog>

  <category name="the category name">
```

```

        <!-- Write the class tags here (code not shown) -->

    </category>

</class_catalog>

```

<category> Write The Categories For The Classes

You organize the classes into categories with the **<category>** tag. The **<category>** tag must be placed within a **<class_catalog>** tag or a **<transaction_catalog>** tag.



For more information about categories, and an example of using the **<category>** tag, refer to **<category> Write The Categories For The Transactions**.

For classes, you can write the **<category>** tag in two ways: named and unnamed.

Put classes into a **<category>** tag when you want to use the class as a transaction input or output parameter or a class data member, and you do not use that class with the Business Interlink types `object_query`, `object_add`, `object_update`, and `object_delete`.

Put classes into a **<category>** tag when you define within the **<driver_settings>** tag one of the Business Interlink types that operate on classes: `object_query`, `object_add`, `object_update`, `object_delete`. Business Interlink Objects of those types can then operate on those classes. When a user, within the Application Designer, uses the Business Interlink Search page to search for the class from which to create that Business Interlink Definition, the user can select a category to list the classes in that category, which were listed in the **<category>** tag in the XML design-time plug-in.

```

<category name="the category name">

    <!-- Write the class tags here (code not shown) -->

</category>

```

You can put classes into an unnamed **<category>** tag when you want to use the class only as a transaction input or output parameter or a class data member.

```

<category>

```

```

        <!-- Write the class tags here (code not shown) -->

    </category>

```



When you place a class into an unnamed **<category>** tag, it will not appear in the Business Interlink Search page in the PeopleSoft Application Designer.

<class> Write A Class

Write a **<class>** tag for each class that your plug-in will use. The **<class>** tag must be placed within a **<category>** tag. Classes are data structures, which consist of members. Members can be basic types such as integers and strings, members can be lists of basic types, and members can be other classes. Classes are used in two ways:

- Application Developers create Business Interlink Definitions/Objects that will perform queries, adds, updates, or deletes upon these classes.
- Classes can be used as members of other classes and as transaction input or output parameters.

Here is an example of how a **<class>** tag is coded for a Freight Carrier plug-in. It creates a class named "Origin".

```

<class name="Origin">

    <member name="Country"

        type="enum(United States,Puerto Rico) "

        default="United States"

        required="true"/>

    <member name="Postal_Code"

        type="string"

        default="94588"

        required="true"/>

</class>

```

<member> Write Members For A Class

Write a **<member>** tag for every data member that a class uses. The **<member>** tag must be placed within a **<class>** tag. A **<member>** tag contains a name and type, and an optional default value.



Whenever you specify a UNIX directory name as a data member, you must end the directory name with a backslash /.

Here is an example of how a **<member>** tag is coded for a Freight Carrier plug-in. It creates an data member named "OriginPostal_Code".

```
<member name="OriginPostal_Code"
    type="string"
    default="94588"
    required="true"/>
```



For more information about the valid values you can use for *type*, see Data Types.

<trans_catalog> Write the Transaction Catalog

In the **<trans_catalog>** tag, write the transaction catalog that you want to use with your plug-in. The transaction catalog provides the names of the transactions, the names of the input and output parameters for each transaction, the data type of each parameter, and optional default values for the input and output parameters.

```
<trans_catalog>
    <category name="the category name">

        <!-- Write the transaction tags here (code not shown) -->

    </category>
</trans_catalog>
```

<category> Write The Categories For The Transactions

Organize the transactions into named **<category>** tags. When a user, within the Application Designer, uses the Business Interlink Search page to search for the transaction from which to create that Business Interlink Definition, the user can select a category to list the transactions in that category, which were listed in the **<category>** tag in the XML design-time plug-in. The **<category>** tag must be placed within a **<transaction_catelog>** tag or a **<class_catelog>** tag.

For example, the following XML code shows a **<category>** tag named PUS transactions.

```
<category name="the category name">

    <!-- Write the transaction tags here (code not shown) -->

</category>
```

<transaction> Write A Transaction

Write a **<transaction>** tag for each transaction that your plug-in will use. A transaction is a named command that can have inputs and outputs. The **<transaction>** tag must be placed within a **<category>** tag.

Here is an example of how a **<transaction>** tag is coded for a Freight Carrier plug-in. It creates a transaction named Calculate Cost.

```
<transaction name="Calculate Cost">

    <input_list>

        <input name="From"

            type="object"

            classname="Origin"/>

        <input name="To"

            type="object"

            classname="Destination"/>

        <input name="Package_Info"

            type="object"

            classname="Package_Information"/>

    </input_list>

    <output_list>
```



```

        <output name="Service_Rate"
            type="list_object"
            classname="Service_Rate"/>
    </output_list>
</transaction>

```

List the Input Parameters <input_list>

If the transaction has input parameters, list each one within an **<input_list>** tag. The **<input_list>** tag must be placed within a **<transaction>** tag.

Write an Input Parameter for a Transaction <input>

Write an **<input>** tag for every input parameter that a transaction uses. The **<input>** tag must be placed within an **<input_list>** tag. An **<input>** tag contains a name and type, and an optional default value.



Whenever you specify a UNIX directory name as an input parameter, you must end the directory name with a backslash /.

Here is an example of how an **<input>** tag is coded for a Freight Carrier plug-in. It creates an input parameter named “From”, which in turn uses the **<class>** named Origin. The data members of Origin are used as input parameters for this transaction.

```

<input name="From"
    type="object"
    classname="Origin"/>

```

Here is an **<input>** tag for an input parameter that is a floating point variable named dollar.

```

<input name="dollar" type="float" default="1.0" required="true"/>

```



For more information about the valid values you can use for *type*, see Data Types.

List the Output Parameters <output_list>

If the transaction has output parameters, list each one within an <output_list> tag. The <output_list> tag must be placed within a <transaction> tag.

Write an Output Parameter for a Transaction <output>

Write an <output> tag for every output parameter that a transaction uses. The <output> tag must be placed within an <output_list> tag.



Whenever you specify a UNIX directory name as an output parameter, you must end the directory name with a backslash /.

Here is an example of how an <output> tag is coded for a Freight Carrier plug-in. It creates an output parameter named "Service_Rate", which in turn uses the <class> named Service_Rate. The data members of Service_Rate are used as output parameters for this transaction.

```
<output name="Service_Rate"
      type="object"
      classname="Service_Rate"/>
```

Here is an <output> tag for an output parameter that is a date variable named overdue.

```
<output name="overdue" type="date" default="12/31/99" />
```



For more information about the valid values you can use for *type*, see Data Types.

The following screen shows, within the Application Designer, the inputs for the Time-in-Transit transaction.

Data Types

The types you can use for configuration parameters, input parameters, output parameters, and data members are:

Type	Definition and format
Int	An integer.
String	A character string.
Float	A floating point variable.
Boolean	A Boolean value of TRUE or FALSE.

Date	A date. The format is YYYY-MM-DD, where YYYY is the year, MM is the month, and DD is the day.
Time	A time. The format is HH:MM:SS, where HH is the hour, MM is the minutes, and SS is the seconds.
Datetime	A date and time. The format is YYYY-MM-DD HH:MM:SS, where YYYY is the year, MM is the month, DD is the day, HH is the hour, MM is the minutes, and SS is the seconds.
Enum	<p>An enumeration. The format is <code>enum(name1, name2,...)</code> where <i>name1</i>, <i>name2</i>, ... are the comma delimited names for this enumeration. Following is an example of an enumeration named Address with the values of Commercial and Residential.</p> <pre><member name="Address" type="enum(Commercial, Residential)"></pre>
Object	<p>A <class> in the <class_catalog> tag. If you set the type as object, you must also provide the classname for that <class>. Following is an example of an object as an input for a transaction; for this example, there must be a <class> named Origin also defined in the XML design-time plug-in.</p> <pre><input name="From" type="object" classname="Origin"/></pre>
Password	A password. This is a character string.
Lists of the above types (not used with configuration parameters)	<p>Each of the above types except for password can be supplied as a list. The list types are list_integer, list_string, list_float, list_boolean, list_date, list_time, list_datetime, and list_object.</p> <p>List types can not be used with relational or logical operators.</p> <p>When you set default values for a list type, you will set it in the same way that you set defaults for the non-list types; you set only one default value in each list.</p>

Writing a XML Design-Time Plug-In using the pshttpenable Runtime Plug-In

Some external systems receive data via HTTP in the form of an XML or HTML request, and then return data in the form of an XML or HTML response. When your external system uses data in this manner, you can use the pshttpenable runtime Plug-in. This means that a Business Interlink runtime plug-in need not be created for your external system.

The pshttpenable runtime plug-in is a self-describing program that encapsulates such external systems. It executes HTTP functions GET and POST for sending XML/HTML requests to the external system, and sets the data document for the response data.

The XML design-time plug-in for the pshttpenable runtime plug-in always describes a transaction. This transaction should define the inputs (also known as request) expected by the API, and the outputs (also known as response) from the API.



For more information about the GET and POST HTTP functions, see <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.

Writing the Configuration Parameters for the pshttpenable runtime plug-in

When you write an XML design-time plug-in for a design-time plug-in that uses the pshttpenable runtime plug-in, you specify some tags within the **<config_parameters>** tag to supply information specific for pshttpenable.

<URL> Specify the pshttpenable Runtime Plug-in

Write the **<URL>** tag as follows to specify the runtime plug-in to be pshttpenable.

```
<URL>file://pshttpenable.dll</URL>
```

Method: Specify the Method Parameters

The Method parameter specifies the HTTP method GET for receiving XML/HTML data from the external system, and the HTTP method POST for sending XML/HTML data to the external system.

Write the **<parameter name = "Method">** tag as follows.

```
<parameter name = "Method"
    type="enum(GET, POST) "
    required="true"
    default = "value"/>
```

where *value* is either GET or POST.

SSL: Specify Secure Link

The SSL parameter specifies if the link to the external system is to be secure.

Write the **<parameter name = "SSL">** tag as follows. YES means secure link, NO means non-secure link.

```
<parameter name = "SSL"
    type="enum(YES,NO) "
    required="true"
    default = "value"/>
```

where *value* is either YES or NO.

Accept_Type: Specify Secure Link

The Accept_Type parameter specifies the Accept type headers.

You can write the **<parameter name = "Accept_Type">** tag as follows. You can also add more types to the enum.

```
<parameter name = "Accept_Type"
    type="enum(xml/html/dhtml) "
    required="true"
    default = "xml/html/dhtml"/>
```

Content_Type: Specify Content Type Headers

The Content_Type parameter specifies the Content type headers.

You can write the **<parameter name = "Content_Type">** tag as follows. You can also add more types to the enum.

```
<parameter name = "Content_Type"

type="enum(Content-Type: text/html,Content-Type: text/xml; charset=utf-
8,Content-Type: application/x-www-form-urlencoded) "

required="true"

default="Content-Type: application/x-www-form-urlencoded" />
```

USERID: Specify User ID

Optional. The USERID parameter can be used when the Method parameter is GET. This is usually used with PASSWORD.

Write the **<parameter name = "USERID">** tag as follows.

```
<parameter name="USERID"

type="string"

required="false"

default="value"/>
```

where *value* is the USERID.

PASSWORD: Specify User Password

Optional. The PASSWORD parameter can be used when the Method parameter is GET. This is usually used with USERID.

Write the **<parameter name = "PASSWORD">** tag as follows.

```
<parameter name="PASSWORD"

type="password"

required="false"

default="value"/>
```

where *value* is the PASSWORD.

MerchantURL: Specify The Merchant URL

The MerchantURL parameter specifies the URL where the merchant API is located. This is where the request will be sent, and from where the response will be received.

Write the **<parameter name = "MerchantURL">** tag as follows.

```
<parameter name="MerchantURL"
    type="string"
    required="true"
    default= "value"/>
```

where *value* is the URL where the merchant API is located.

InDataType: Specify Input Data Type

The InDataType parameter specifies the type of input data that is expected by the merchant API.

Write the **<parameter name = "InDataType">** tag as follows.

```
<parameter name="InDataType"
    type="enum(XML, HTML, DHTML) "
    required="true"
    default="value" />
```

where *value* is XML, HTML, DHTML, NAMEVALUEPAIR, or XMLNAMEVALUEPAIR.

OutDataType: Specify Output Data Type

The OutDataType parameter specifies the type of output data that is expected by the merchant API.

Write the **<parameter name = "OutDataType">** tag as follows.

```
<parameter name="OutDataType"
    type="enum(XML, HTML, DHTML) "
    required="true"
    default = "value"/>
```

where *value* is either XML, HTML, or DHTML.

XML_Validation: Check for Errors in XML Syntax

The XML_Validation parameter causes the XML parsers to check for errors in the XML syntax of the XML design-time plug-in.

Write the **<parameter name = "XML_Validation">** tag as follows.

```
<parameter name="XML_Validation"
  type="bool"
  required="true"
  default="value" />
```

where *value* is either TRUE or FALSE.

Input_File_Name: Debug Input Data

Optional. The Input_File_Name parameter is a debugging tool. It points to an XML, HTML, or DHTML file containing input data. This file is sent as the request for the transaction, overriding the data that is defined in the XML Design-time plug-in.

Write the **<parameter name = "Input_File_Name">** tag as follows.

```
<parameter name="Input_File_Name"
  type="string"
  required="false"
  default="file_name"/>
```

where *file_name* is the file name and path.

Request_File_Name: Debug Request Data

Optional. The Request_File_Name parameter is a debugging tool. It points to a location where the request will be written as a text file to the specified location.

Write the **<parameter name = "Request_File_Name">** tag as follows.

```
<parameter name="Request_File_Name"
  type="string"
  required="false"
  default="file_name"/>
```

where *file_name* is the file name and path.

Simulated_Response_File_Name: Debug Request Data

Optional. The `Simulated_Response_File_Name` parameter is a debugging tool. It points to an XML, HTML, or DHTML file containing output data. This file is used as the response for the transaction, overriding the response that is sent from the `MerchantURL`.

Write the `<parameter name = "Simulated_Response_File_Name">` tag as follows.

```
<parameter name="Simulated_Response_File_Name"
  type="string"
  required="false"
  default="file_name"/>
```

where *file_name* is the file name and path.

Response_File_Name: Debug Request Data

Optional. The `Response_File_Name` parameter is a debugging tool. It points to a location where the response will be written as a text file to the specified location.

Write the `<parameter name = "Response_File_Name">` tag as follows.

```
<parameter name="Response_File_Name"
  type="string"
  required="false"
  default="file_name"/>
```

where *file_name* is the file name and path.

Metatag: Debug Request Data

The `Metatag` parameter allows you to create a DOCTYPE statement if the merchant API requires it.

An example of how you could write the `<parameter name = "Metatag">` tag follows.

```
<parameter name="Metatag"
  type="string"
  required="false"
  default="
    "<!DOCTYPE postreq SYSTEM %34file:/peoplesoft/peoplesoft.dtd%34> " />
```

Writing the pshttpenable Runtime Plug-in Input Parameters

You write the input parameters to match the inputs for the transaction that is provided by the merchant.



For more information on writing input parameters, see Write an Input Parameter for a Transaction <input>.

Writing the pshttpenable Runtime Plug-in Output Parameters

You write the output parameters to match the outputs for the transaction that is provided by the merchant.



For more information on writing output parameters, see Write an Output Parameter for a Transaction <output>.

Writing the pshttpenable Runtime Plug-in Classes

You can define classes to be used as data structures for the inputs or outputs.



For more information on writing classes, see <class_catalog> Write the Class Catalog.

Safe Characters

Within name-value pairs, the following characters are safe characters:

- 0-9
- A-Z
- a-z
- ? & (blank space)

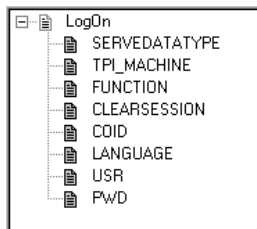
If the application developer uses any other characters, they must encode them for HTML.

Example of XML Design-Time Plug-ins using pshttppenable

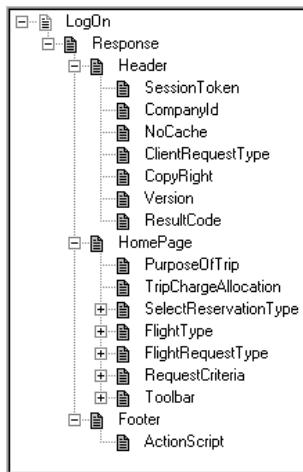
This example shows an example of an XML design-time plug-in. This plug-in uses the GET method.

Diagram of the Example Transaction

Here is a diagram of the Sabre LogOn transaction resulting from the XML code in this chapter, showing the input and output structure.



Sabre LogOn Transaction Inputs



Sabre LogOn Transaction Outputs

The following is the XML design-time plug-in.

```
<?xml version="1.0" ?>

<interface_driver>

  <general_info>

    <description>Sabre Prototype Plug-in</description>

    <version>1</version>
```

```

    <lastupdate>12/3/99</lastupdate>

    <comments></comments>

</general_info>

<driver_settings>

    <option type="static_catalog" supported="true"/>

    <option type="transaction" supported="true"/>

    <option type="input_class_expandable" supported="true"/>

    <option type="output_class_expandable" supported="true"/>

    <option type="hierarchical_model" supported="true"/>

    <relational_op>

</relational_op>

    <logical_op>

</logical_op>

</driver_settings>

<config_parameters>

    <!-- Set the URL to the Httpenable runtime plug-in. -->

    <URL>file://Httpenable.dll</URL>

    <!-- Set the method to GET. -->

    <parameter name="Method"

        type="enum(GET, POST) "

        required="true"

        default="GET"/>

    <!-- Set the USERID and PASSWORD. -->

    <parameter name="USERID" type="string" required="false" />

    <parameter name="PASSWORD" type="password" required="false" />

```

```
<!-- Specify the location of the merchant API. -->

<parameter name="MerchantURL" type="string" required="true"
    default="http://peoplesoft.sabre.com/lcgi-bin/bts.tpi"/>

<!-- Specify the input and output data expected by the -->
<!-- merchant API. In this case, the data type is XML. -->
<parameter name="InDataType" type="enum(XML, HTML, DHTML)"
    required="true" default="XML" />
<parameter name="OutDataType" type="enum(XML, HTML, DHTML)"
    required="true" default="XML" />

<!-- Set XML validation to FALSE. -->
<parameter name="XML_Validation" type="bool"
    required="true" default="FALSE" />

<!-- Set up files for debugging. -->
<!-- Send the request to DebugQueryString.txt. -->
<!-- Send the response to DebugResponse.txt. -->
<parameter name="Input_File_Name" type="string"
    required="false" default=""/>
<parameter name="Request_File_Name" type="string"
    required="false" default="c:\sabre\DebugQueryString.txt"/>
<parameter name="Simulated_Response_File_Name" type="string"
    required="false" default=""/>
<parameter name="Response_File_Name" type="string"
    required="false" default="c:\sabre\DebugResponse.txt"/>
<parameter name="Metatag" type="string" required="false"
    default=""/>
```

```
</config_parameters>

<class_catalog>

<category name="SabreLogin">

    <!-- This class is used in the Response class. -->

    <class name="HeaderLogin">

        <member name="SessionToken" type="string"

            attribute="xml_attr" default = ""/>

        <member name="CompanyId" type="string"

            default="" required="true"/>

        <member name="NoCache" type="string"

            default="NA" required="true"/>

        <member name="ClientRequestType" type="string"

            default="NA" required="true"/>

        <member name="CopyRight" type="string"

            default="NA" required="true"/>

        <member name="Version" type="string"

            default="NA" required="true"/>

        <member name="ResultCode" type="string"

            default="NA" required="true"/>

    </class>

    <!-- This class is used in the Response class. -->

    <class name="HomePage">

        <member name="PurposeOfTrip" type="string"

            default="" required="false"/>

        <member name="TripChargeAllocation" type="string"
```

```

        default="" required="false"/>

<member name="SelectReservationType" type="object"

        classname ="SelectReservationType" required ="false"/>

<member name="FlightType" type="object"

        classname ="FlightType" required ="false"/>

<member name="FlightRequestType" type="object"

        classname ="FlightRequestType" required ="false"/>

<member name="RequestCriteria" type="object"

        classname ="RequestCriteria" required ="false"/>

<member name="Toolbar" type="object"

        classname ="Toolbar" required ="false"/>

</class>

<!-- This class is used in the Response class. -->

<class name="Footer">

    <member name="ActionScript" type="string"

        default="" required="false"/>

</class>

<!-- Response is the class used as the output parameter. -->

<!-- See the output_list tag in this example. -->

<class name="Response">

    <member name="Header" type="object"

        classname="HeaderLogin" default="" required="false"/>

    <member name="HomePage" type="object"

        classname="HomePage" default="" required="false"/>

    <member name="Footer" type="object"

        classname="Footer" default="" required="false"/>

</class>

```



```
</category>
```

```
</class_catalog>
```

```
<!-- The following classes have been edited out of this example to make the
example shorter: FlightType, SelectReservationType, FlightRequestType,
FlightType, RequestCriteria, Destination, ArrivalDepartureInfo,
TripCityTimeInfo, CodeDescription, Date1, Time1, Toolbar, Button,
LoginInformation -->
```

```
<trans_catalog>
```

```
  <category name="Sabre transactions">
```

```
    <!-- Write the inputs for the LogOn transaction. -->
```

```
    <transaction name="LogOn">
```

```
      <input_list>
```

```
        <input name="SERVEDATATYPE" type="string"
```

```
          default="SERVE_XML" required="false"/>
```

```
        <input name="TPI_MACHINE" type="string"
```

```
          default="http://peoplesoft.sabre.com"
```

```
          required="false"/>
```

```
        <input name="FUNCTION" type="string"
```

```
          default="TPIScriptLogin" required="false"/>
```

```
        <input name="CLEARSESSION" type="string" default="YES"
```

```
          required="false"/>
```

```
        <input name="COID" type="string" default="PEOPLESFT"
```

```
          required="false"/>
```

```
        <input name="LANGUAGE" type="string" default="en_US"
```

```
          required="false"/>
```

```
        <input name="USR" type="string" default="ROCKY"
```

```

        required="false"/>

        <input name="PWD" type="string" default="bts"

        required="false"/>

    </input_list>

    <!-- Write the outputs for the LogOn transaction. -->

    <!--The outputs consist of one class: Response. -->

    <output_list>

        <output name="Response" type="object"

        classname="Response"/>

    </output_list>

</transaction>

</category>

</trans_catalog>

</interface_driver>

```

pshttpenable Runtime Plug-in: Reading an XML String

This section defines the requirements for a generic XML design-time plug-in that is designed to read in a XML string representing data. Executing the plug-in results in a Business Interlink document populated with the data from the XML string. You can then use that Business Interlink document and its data in your PeopleCode program.

1. Write (or use a previously written) XML design-time plug-in.

The input in this XML design-time plug-in must consist of only one input parameter of string type. The output parameters in the XML design-time plug-in must fit the XML string that will be passed as input in the PeopleCode program.

2. From the PeopleSoft Application Designer, open a Business Interlink using this XML design-time plug-in, and save it as a Business Interlink Definition. The name you save it under must have "IMPORTXML" as its first nine letters.
3. Write a PeopleCode program that uses this Business Interlink Definition.

This PeopleCode program has, as input, a string that is structured to fill the output parameters

with data. Use the BIDocs method AddValue to read this input into the Business Interlink document, and use the BIDocs methods GetDoc and GetValue to get the data from the Business Interlink document.



For more information about creating a Business Interlink Definition, see the *PeopleSoft Business Interlink Runtime Plug-in Programming Guide*, Designing a Business Interlink Definition. For more information about writing the PeopleCode program, see Running the Business Interlink Object. For more information about the BIDocs (Business Interlink document) methods and the Business Interlink methods, see Using the Business Interlink Object Methods.

Example of Design-Time Plug-in for Reading XML String

Here is an example of an XML design-time plug-in for reading an XML string.

```
<?xml version="1.0" ?>

<interface_driver>

  <general_info>

    <description>ImportXML Prototype Plug-in</description>

    <version>1</version>

    <lastupdate>01/28/00</lastupdate>

    <comments></comments>

  </general_info>

  <driver_settings>

    <option type="static_catalog" supported="true"/>

    <option type="transaction" supported="true"/>

    <option type="input_class_expandable" supported="true"/>

    <option type="output_class_expandable" supported="true"/>

    <option type="hierarchical_model" supported="true"/>

  </driver_settings>

  <relational_op>

  </relational_op>

  <logical_op>
```

```

    </logical_op>

</driver_settings>

<!--Set the configuration parameters as follows. -->

<config_parameters>

    <URL>file://Httpenable.dll</URL>

    <parameter name="Method" type="enum(GET, POST) "

        required="true" default="POST"/>

    <parameter name="SSL" type="enum(YES,NO) " required="true"

        default="NO" />

    <parameter name="Accept_Type" type="enum(text/xml/html) "

        required="true" default="text/xml/html" />

    <parameter name="Content_Type"

        type="enum(Content-Type: text/html,Content-Type: text/xml; charset=utf-
8,Content-Type: application/x-www-form-urlencoded) "

        required="true"

        default="Content-Type: application/x-www-form-urlencoded" />

    <parameter name="USERID" type="string" required="false" />

    <parameter name="PASSWORD" type="password" required="false" />

    <parameter name="MerchantURL" type="string"

        required="true" default=""/>

    <parameter name="InDataType"

        type="enum(XML, HTML, DHTML) "

        required="true" default="XML" />

    <parameter name="XML_As_Name_Value_Pair" type="enum(YES,NO) "

        required="true" default="NO" />

    <parameter name="OutDataType" type="enum(XML, HTML, DHTML) "

        required="true" default="XML" />

    <parameter name="XML_Validation" type="bool" required="true"

        default="FALSE" />

```

```

<parameter name="Input_File_Name" type="string"
    required="false" default=""/>
<parameter name="Request_File_Name" type="string"
    required="false" default="c:\temp\DebugQueryString.txt"/>
<parameter name="Simulated_Response_File_Name" type="string"
    required="false" default=""/>
<parameter name="Response_File_Name" type="string"
    required="false" default="c:\temp\DebugResponse.txt"/>
<parameter name="Metatag" type="string" required="false"
    default="" />
</config_parameters>

<class_catalog>
<category name="SkillsVillage_Classes">

    <class name="postreqresponse">
        <member name="error" type="object"
            classname="error" default =""/>
        <member name="candidates" type="object"
            classname="candidates" default =""/>
    </class>

    <class name="error">
        <member name="errorcode" type="int" default="1"
            required="false"/>
        <member name="errortext" type="string" default=""
            required="false"/>
    </class>

```

```
<class name="candidates">

  <member name="user" type="list_object" classname="users"

    default = ""/>

</class>

<class name="users">

  <member name="userid" type="string"

    default="" required="false" />

  <member name="username" type="string"

    default="" required="false" />

  <member name="comments" type="string"

    default="" required="false" />

  <member name="location" type="object"

    classname="responselocation" default = ""/>

  <member name="desiredloc" type="string"

    default="" required="false" />

  <member name="role" type="string"

    default="" required="false"/>

  <member name="availability" type="string"

    default="" required="false"/>

  <member name="rate" type="float"

    default="" required="false"/>

  <member name="relevancy" type="int"

    default="" required="false"/>

  <member name="rating" type="int"

    default="" required="false"/>

</class>

<class name="responselocation">

  <member name="city" type="string"
```

```

        default = "" required="false"/>

<member name="state" type="string"

        default = "" required="false"/>

<member name="zip" type="string"

        default = "" required="false"/>

<member name="country" type="string"

        default = "" required="false"/>

</class>

</category>

</class_catalog>

<trans_catalog>

  <category name="ImportXML_Transactions">

    <transaction name="ImportXML">

      <input_list>

        <!-- XMLString matches the name of the PeopleCode input. -->

        <input name="XMLString" type="string"

          default="" required="true"/>

      </input_list>

      <output_list>

        <output name="postreqresponse" type="object"

          classname="postreqresponse" required="false"/>

      </output_list>

    </transaction>

  </category>

</trans_catalog>

```

```
</interface_driver>
```

Example of PeopleCode for Reading XML String

This example shows the PeopleCode that corresponds to the ImportXML example. The importXMLString variable is a character string containing data that fills the output for the ImportXML design-time plug-in. If you examine the importXMLString variable in the following listing, you can see that its structure matches the structure of the Example of Design-Time Plug-in for Reading XML String.



For more information on writing a PeopleCode program for Business Interlinks, refer to Running the Business Interlink Object.

```
Local string &importXMLString;

Local Interlink &ImportXML;

Local BIDocs &InputDoc, &OutputDoc;

Local number &EXECSRSLT;

Local number &ret;

Local BIDocs &zipdoc;

Local BIDocs &user;

Local string &locValue;

Local string &userid;

&ImportXML = GetInterlink(Interlink.IMPORTXML);

&InputDoc = &ImportXML.GetInputDocs("");

&importXMLString = "<?xml
version='1.0'?'><postreqresponse><error><errorcode>1</errorcode><errortext></er
rortext></error><candidates><user><userid>9719</userid><username>KWoOLF</usernam
e><comments></comments><location><city>Dallas</city><state>TX</state><zip>75240<
/zip><country>USA</country></location><desiredloc>US-Texas</desiredloc><role>No
Preference</role><availability>2000-02-
01</availability><rate>35.0</rate><relevancy>10</relevancy><rating>0</rating></u
ser><user><userid>10411</userid><username>wlbond008</username><comments>Willing
to take just about any length
assignment.</comments><location><city>Roanoke</city><state>VA</state><zip>24017<
/zip><country>USA</country></location><desiredloc>No
Preference</desiredloc><role>No Preference</role><availability>2000-02-
13</availability><rate>35.0</rate><relevancy>10</relevancy><rating>0</rating></u
ser></candidates></postreqresponse>";
```



```

/* --- XMLString is the name of the single input. ---*/

&ret = &InputDoc.AddValue("XMLString", &importXMLString);

&EXECSRSLT = &ImportXML.Execute();

/* The Business Interlink document is now filled with the data from the input
string. */

If (&EXECSRSLT <> 1) Then

    /* The instance failed to execute */

    WinMessage("Error occurred during processing BI.");

Else

    &OutputDoc = &ImportXML.GetOutputDocs("");

    /* Extract the user data from the Business Interlink document. */

    &user = &OutputDoc.GetDoc("postregresponse.candidates.user");

    If (&user.GetStatus() = 0) Then

        &count = &user.GetCount("user");

        &i = 1;

        /* --- user is an object_list in the ImportXML design-time */
        /* plug-in, so a loop is needed to extract its output values. */

        While (&i <= &count)

            &locdoc = &user.GetDoc("location");

            &ret = &user.GetValue("userid", &userid);

            &ret = &locdoc.GetValue("zip", &zipValue);

            &ret = &user.GetNextDoc();

            &i = &i + 1;

        End-While;

    End-If;

End-If;

```

XML Template for pshttpenable Runtime Plug-in

You can use the following code as a template when you write an XML design-time plug-in for the pshttpenable Runtime Plug-in.

```
<?xml version="1.0"?>

<interface_driver>

  <general_info>

    <description>HttpEnable Prototype</description>

    <version>1</version>

    <lastupdate>August 2000</lastupdate>

    <comments>Please insert your details for this Business Interlink
design</comments>

  </general_info>

  <driver_settings>

    <option type="static_catalog" supported="true"/>

    <option type="transaction" supported="true"/>

    <option type="input_class_expandable" supported="true"/>

    <option type="output_class_expandable" supported="true"/>

    <option type="hierarchical_model" supported="true"/>

    <relational_op/>

    <logical_op/>

  </driver_settings>

  <config_parameters>

    <URL>file://pshttpenable.dll</URL>

    <parameter name="Method" type="enum(GET,POST)" required="true"
default="POST"/>

    <parameter name="SSL" type="enum(YES,NO)" required="true"
default="NO" />

    <parameter name="Accept_Type" type="enum(text/xml/html)"
required="true" default="text/xml/html"/>

    <parameter name="Content_Type" type="enum(Content-Type:
text/html,Content-Type: text/xml; charset=utf-8,Content-Type: application/x-www-
form-urlencoded)" required="true" default="Content-Type: application/x-www-form-
urlencoded"/>
```

```

        <parameter name="MerchantURL" type="string" required="true"
default="" />

        <parameter name="InDataType"
type="enum (XML,HTML,DHTML,NAMEVALUEPAIR,XMLNAMEVALUEPAIR) " required="true"
default="XML" />

        <parameter name="OutDataType" type="enum (XML,HTML,DHTML) "
required="true" default="XML" />

        <parameter name="XML_Validation" type="bool" required="true"
default="FALSE" />

        <parameter name="Metatag" type="string" required="true" default="" />

        <parameter name="Input_File_Name" type="string" required="false"
default="" />

        <parameter name="Simulated_Response_File_Name" type="string"
required="false" default="" />

        <parameter name="Response_File_Name" type="string" required="false"
default="c:\temp\DebugResponse.txt" />

        <parameter name="Request_File_Name" type="string" required="false"
default="c:\temp\DebugRequest.txt" />

    </config_parameters>

    <class_catalog>

        <category name="My Classes Category">

        </category>

    </class_catalog>

    <trans_catalog>

        <category name="My Transactions Category">

            <transaction name="My Transaction">

                <input_list>

                    <input name="MyRequest" type="string"
default="" />

                </input_list>

                <output_list>

```

```
                                <output name="MyResponse" type="string"
required="true"/>
                                </output_list>
                            </transaction>

                        </category>
                    </trans_catalog>
</interface_driver>
```

CHAPTER 5

Deploying your XML Design-Time Plug-In

Once you have written your XML design-time plug-in, place it into the following directory:

```
PSHome\bin\client\winx86\interfacedrivers
```

Where *PSHome* is the directory where PeopleTools is installed.

Index

A

Accept_Type 4-2
Architecture 1-1

B

BIDocs objects
 checking 3-14
BIDocValidate XML tag 3-14
Business Classes
 see classes 2-2
Business Interlink
 actions when running 1-2
 runtime plug-in location 3-15
 runtime plug-in method used to locate 3-16
Business Interlink Architecture 1-1
Business Interlink Object
 actions taken to create 1-4
Business Interlink type
 adding 3-11
 list of 3-11
Business Transactions
 see transactions 2-4

C

catalog
 class 3-16
 transaction 3-19
category XML tag 3-17, 3-20
CD-ROM
 ordering ii
class
 adding a data member 3-19
 categories 3-17
 writing 3-18
 writing catalog 3-16
class XML tag 3-18
class_catalog XML tag 3-16
classes
 initial design 2-2
config_parameters XML tag 3-13
configuration parameters
 adding 3-13
 data types 3-22
 initial design 2-1
Content_Type 4-3

D

data member
 adding to a class 3-19
data types 3-22
deploying the XML design-time plug-in 5-1
description
 creating 3-9
description XML tag 3-9
design
 initial 2-1
DOCTYPE
 setting for pshttpenable runtime plug-in 4-6
driver_settings XML tag 3-10

G

general_info XML tag 3-9
GET
 setting for HTTP 4-2
 setting password 4-3
 setting userid 4-3
graphic
 adding 3-10

H

HTTP method
 setting GET and POST 4-2
 setting security 4-2

I

image XML tag 3-10
InDataType 4-4
input data
 debugging with pshttpenable runtime plug-in 4-5
input parameter
 adding to a transaction 3-21
input parameters
 data types 3-22
input XML tag 3-21
Input_File_Name 4-5
input_list XML tag 3-21
interface_driver 3-9

L

logical operator 3-12
logical_op XML tag 3-12

M

member XML tag 3-19
merchant API
 setting for pshtpenable runtime plug-in 4-4
 setting input data type 4-4
 setting output data type 4-4
MerchantURL 4-4
Metatag 4-6
Method 4-2

O

operator XML tag 3-13
option XML tag 3-11
OutDataType 4-4
output parameter
 adding to a transaction 3-22
output parameters
 data types 3-22
output XML tag 3-22
output_list XML tag 3-22

P

parameter XML tag 3-14
 Accept_Type setting 4-2
 Content_Type setting 4-3
 Input_File_Name setting 4-5
 InputDataType setting 4-4
 MerchantURL setting 4-4
 Method setting 4-2
 OutputDataType setting 4-4
 PASSWORD setting 4-3
 Request_File_Name setting 4-5
 Response_Name setting 4-6
 Simulated_Response_Name setting 4-6
 SSL setting 4-2
 USERID setting 4-3
 XML_Validation setting 4-5
PASSWORD 4-3
PeopleBooks
 CD-ROM, ordering ii
 printed, ordering ii
PepperCode
 checking BIDocs objects 3-14
POST
 setting for HTTP 4-2
pshtpenable runtime plug-in
 parameter name = Accept_Type 4-2

 parameter name = Content_Type 4-3
 parameter name = Input_File_Name 4-5
 parameter name = InputDataType 4-4
 parameter name = MerchantURL 4-4
 parameter name = Metatag 4-6
 parameter name = Method 4-2
 parameter name = OutputDataType 4-4
 parameter name = PASSWORD 4-3
 parameter name = Request_File_Name 4-5
 parameter name = Response_Name 4-6
 parameter name = Simulated_Response_Name 4-6
 parameter name = SSL 4-2
 parameter name = USERID 4-3
 parameter name = XML_Validation 4-5
URL XML tag 4-1

R

relational operator 3-12
relational_op XML tag 3-12
request
 debugging with pshtpenable runtime plug-in 4-5
Request_File_Name 4-5
response
 debugging with pshtpenable runtime plug-in 4-6
Response_Name 4-6
runtime plug-in
 location 3-15
 method used to locate 3-16

S

secure link
 setting for HTTP 4-2
Simulated_Response_Name 4-6
SSL 4-2

T

trans_catalog XML tag 3-19
transaction
 adding an input parameter 3-21
 adding an output parameter 3-22
 catagories 3-20
 input list 3-21
 output list 3-22
 writing 3-20
 writing catalog 3-19
transaction XML tag 3-20
transactions
 initial design 2-4

U

URL XML tag 3-15
 setting for pshttpenable runtime plug-in 4-1
 USERID 4-3

V

version number
 creating 3-9
 version XML tag 3-9

X

XML design-time plug-in
 deploying 5-1
 example 3-2
 template 3-1
 XML tag
 BIDocValidate 3-14
 category 3-17, 3-20
 class 3-18
 class_catalog 3-16
 config_parameters 3-13
 description 3-9
 driver_settings 3-10
 general_info 3-9
 image 3-10
 input 3-21

input_list 3-21
 interface_driver 3-9
 logical_op 3-12
 member 3-19
 operator 3-13
 option 3-11
 output 3-22
 output_list 3-22
 parameter 3-14
 parameter name = Accept_Type 4-2
 parameter name = Content_Type 4-3
 parameter name = Input_File_Name 4-5
 parameter name = InputDataType 4-4
 parameter name = MerchantURL 4-4
 parameter name = Metatag 4-6
 parameter name = Method 4-2
 parameter name = OutputDataType 4-4
 parameter name = PASSWORD 4-3
 parameter name = Request_File_Name 4-5
 parameter name = Response_Name 4-6
 parameter name = Simulated_Response_Name 4-6
 parameter name = SSL 4-2
 parameter name = USERID 4-3
 parameter name = XML_Validation 4-5
 relational_op 3-12
 trans_catalog 3-19
 transaction 3-20
 URL 3-15
 URL setting for pshttpenable runtime plug-in 4-1
 version 3-9
 XML_Validation 4-5

