PeopleSoft®

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Business Interlinks

June 2004

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Business Interlinks
SKU PT845BIS-B 0604

Open Source Disclosure

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

Apache Software Foundation

This product includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright (c) 1999-2000 The Apache Software Foundation. All rights reserved.

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

OpenSSL

Copyright (c) 1998-2003 The OpenSSL Project. All rights reserved.

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

SSLeay

Copyright (c) 1995-1998 Eric Young. All rights reserved.

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

Loki Library

Copyright (c) 2001 by Andrei Alexandrescu. This code accompanies the book:

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

Contents

Part 2
Business Interlinks for Design-Time Plug-In Programming

Part 3
Business Interlinks for Runtime Plug-in Programming

About This PeopleBook

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

This preface discusses:

- PeopleSoft application prerequisites.
- PeopleSoft application fundamentals.
- Related documentation.
- Typographical conventions and visual cues.
- Comments and suggestions.
- Common elements in PeopleBooks.

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

PeopleSoft Application Prerequisites

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

See *Enterprise PeopleTools 8.45 PeopleBook: Using PeopleSoft Applications*.

You might also want to complete at least one PeopleSoft introductory training course.

You should be familiar with navigating the system and adding, updating, and deleting information by 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.

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

PeopleSoft Application Fundamentals

Each application PeopleBook provides implementation and processing information for your PeopleSoft database. However, additional, essential information describing the setup and design of your system appears in a companion volume of documentation called the application fundamentals PeopleBook. Each PeopleSoft product line has its own version of this documentation.

The application fundamentals PeopleBook consists of important topics that apply to many or all PeopleSoft applications across a product line. Whether you are implementing a single application, some combination of applications within the product line, or the entire product line, you should be familiar with the contents of this central PeopleBook. It is the starting point for fundamentals, such as setting up control tables and administering security.

Related Documentation

This section discusses how to:

- Obtain documentation updates.
- Order printed documentation.

Obtaining Documentation Updates

You can find updates and additional documentation for this release, as well as previous releases, on the PeopleSoft Customer Connection website. Through the Documentation section of PeopleSoft 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 that is delivered on your PeopleBooks CD-ROM.

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

See Also

PeopleSoft Customer Connection, https://www.peoplesoft.com/corp/en/login.jsp

Ordering Printed Documentation

You can order printed, bound volumes of the complete PeopleSoft documentation that is delivered on your PeopleBooks CD-ROM. PeopleSoft makes printed documentation available for each major release shortly after the software is shipped. Customers and partners can order printed PeopleSoft documentation by using any of these methods:

- Web
- Telephone
- Email

Web

From the Documentation section of the PeopleSoft Customer Connection website, access the PeopleBooks Press website under the Ordering PeopleBooks topic. The PeopleBooks Press website is a joint venture between PeopleSoft and MMA Partners, the book print vendor. Use a credit card, money order, cashier's check, or purchase order to place your order.

Telephone

Contact MMA Partners at 877 588 2525.

Email

Send email to MMA Partners at peoplesoftpress@mmapartner.com.

See Also

PeopleSoft Customer Connection, https://www.peoplesoft.com/corp/en/login.jsp

Typographical Conventions and Visual Cues

This section discusses:

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

Typographical Conventions

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

Typographical Convention or Visual Cue	Description
Bold	Indicates PeopleCode function names, method names, language constructs, and PeopleCode reserved words that must be included literally in the function call.
Italics	Indicates field values, emphasis, and PeopleSoft or other book-length publication titles. In PeopleCode syntax, italic items are placeholders for arguments that your program must supply. We also use italics when we refer to words as words or letters as letters, as in the following: Enter the letter *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 the W key.
`Monospace font`	Indicates a PeopleCode program or other code example.
" " (quotation marks)	Indicate chapter titles in cross-references and words that are used differently from their intended meanings.
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe (|).

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

Visual Cues

PeopleBooks contain the following visual cues.

Notes

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

Note. Example of a note.

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

Important! Example of an important note.

Warnings

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

Warning! Example of a warning.

Cross-References

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

Country, Region, and Industry Identifiers

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

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

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

Country Identifiers

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

See Appendix A, “ISO Country and Currency Codes,” ISO Country Codes.

Region Identifiers

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

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

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

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

Currency Codes

Monetary amounts are identified by the ISO currency code.

See Appendix A, “ISO Country and Currency Codes,” ISO Currency Codes.

Comments and Suggestions

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

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

Or send email comments to doc@peoplesoft.com.

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

Common Elements in These PeopleBooks

As of Date — The last date for which a report or process includes data.

Business Unit — An ID that represents a high-level organization of business information. You can use a business unit to define regional or departmental units within a larger organization.

Description — Enter up to 30 characters of text.

Effective Date — The date on which a table row becomes effective; the date that an action begins. For example, to close out a ledger on June 30, the effective date for the ledger closing would be July 1. This date also determines when

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

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Process Scheduler

Enterprise PeopleTools 8.45 PeopleBook: Using PeopleSoft Applications

PeopleSoft Business Interlinks

This PeopleBook describes PeopleSoft Business Interlinks.

Note. PeopleSoft Business Interlinks is a deprecated product. Support will be maintained for this product, but no new development will be produced for Business Interlinks and documentation for Business Interlinks will not be published in future releases beginning with PeopleTools 8.45. PeopleSoft advises that you use PeopleSoft Integration Broker's sync request and response functionality in place of Business Interlinks.

PeopleSoft Business Interlinks Overview

This PeopleBook covers the concepts of Business Interlinks in four distinct parts:

- Business Interlinks for Application Developers
- Business Interlinks for Design-Time Plug-In Programming
- Business Interlinks for Runtime Plug-In Programming
- Business Interlinks for XML

The "About These PeopleBooks" preface contains general product line information, such as related documentation, common page elements, and typographical conventions. This preface also contains a glossary with useful terms that are used in PeopleBooks.

PART 1

Business Interlinks for Application Developers

CHAPTER 1

Getting Started with PeopleSoft Business Interlinks

This chapter discusses:

- PeopleSoft Business Links overview
- Implementing PeopleSoft Business Interlinks

Note. Business Interlinks is a deprecated product. Support will be maintained for this product, but no new development will be produced for Business Interlinks and documentation for Business Interlinks will not be published in future releases beginning with PeopleTools 8.45. PeopleSoft advises that you use PeopleSoft Integration Broker's synchronous request and response functionality to replace Business Interlinks.

See Also

Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Integration Broker

PeopleSoft Business Interlinks Overview

Business Interlinks enable you to perform component-based, real-time 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.

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.

Business Interlinks are created in PeopleSoft Application Designer, so you should ensure that you are familiar with PeopleTools and PeopleSoft Application Designer.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleSoft Application Designer*.

Implementing Business Interlinks

The functionality to create business interlinks for your applications is delivered as part of standard PeopleSoft PeopleTools that are provided with all PeopleSoft products.

Several activities must be completed before you begin to create business interlinks for your implementation.

- Install your PeopleSoft application according to the installation guide for your database type.

See The PeopleSoft Installation Guide for your platform and product line.

- Establish a user profile that gives you access to PeopleSoft Application Designer and any other processes you will use.

  See *Enterprise PeopleTools 8.45 PeopleBook: Security Administration*.

CHAPTER 2

Understanding Business Interlinks

This chapter provides an overview of Business Interlinks and discusses:

- Business Interlink architecture.
- Business Interlink actions.
- Business Interlink creation.
- An example of how Business Interlinks can be used.
- An example of an XML design-time plug-in.
- Common Business Interlink terms.

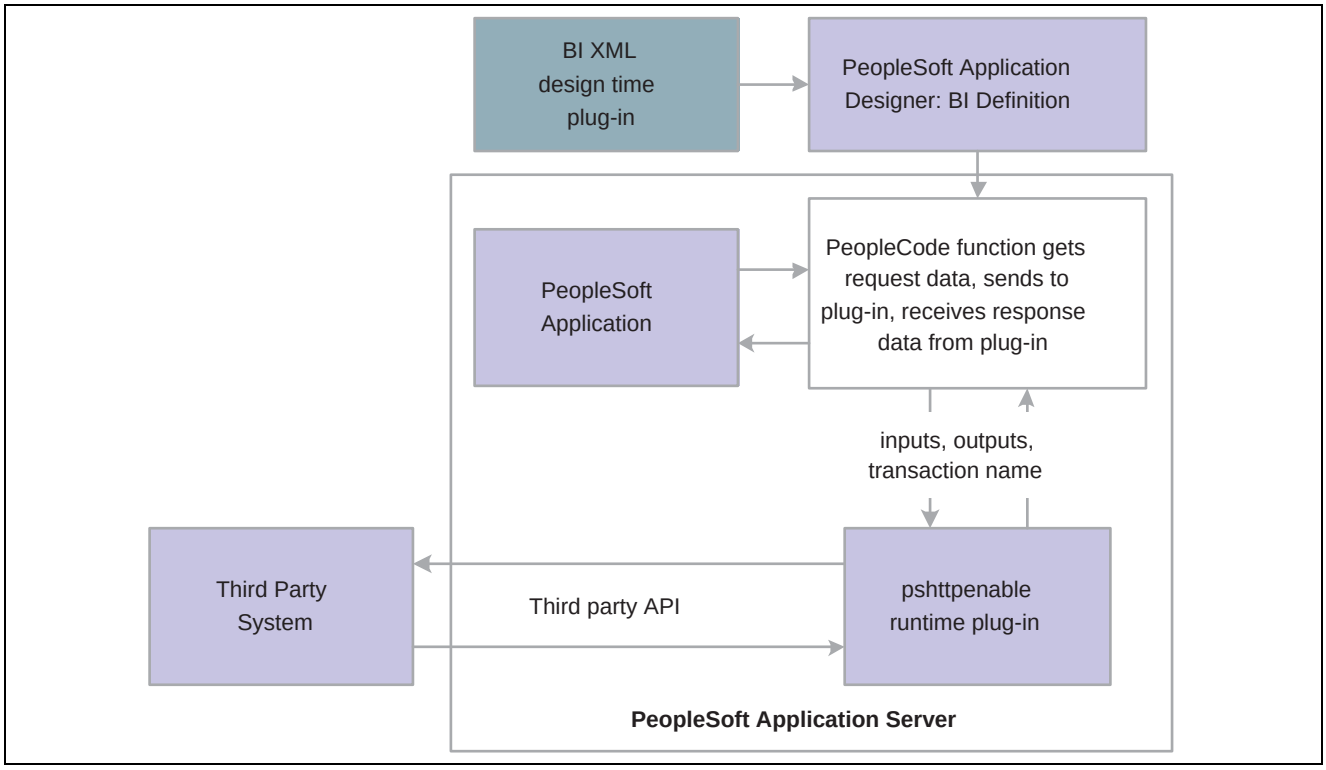
Business Interlink Architecture

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.

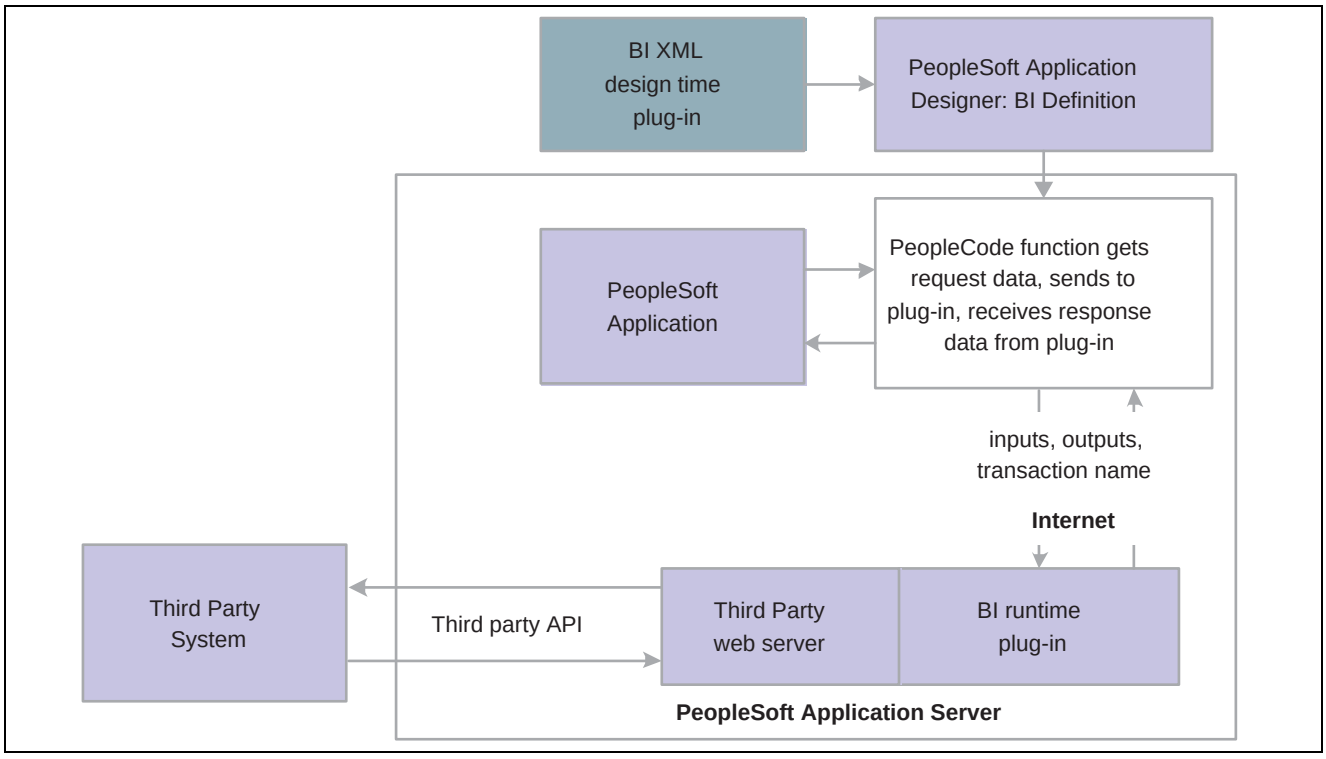
When a business event triggers the execution of a Business Interlink, the Component Processor synchronously calls the Business Interlink processor, which in turn invokes the appropriate Business Interlink plug-in. The plug-in provides a wrapper around the third party API and is designed to support any 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.

Business Interlink Actions

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



Using the pshttpenable runtime plug-in



Using any runtime plug-in

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 click a button labeled *Calculate.*

2. The triggered event (clicking 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 though 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.
6. 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 & Cost, the output values could then be displayed on that page.

Business Interlink Creation

To allow users to create and run Business Interlinks, developers must perform the following tasks.

1. A developer designs the shape of a transaction(s) (inputs, outputs, name). For example, the action could be to tell PScustomer 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 XML.
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 form 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 PScustomer 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 PScustomer and calculate shipping costs.

Example of Calculating Shipping Costs

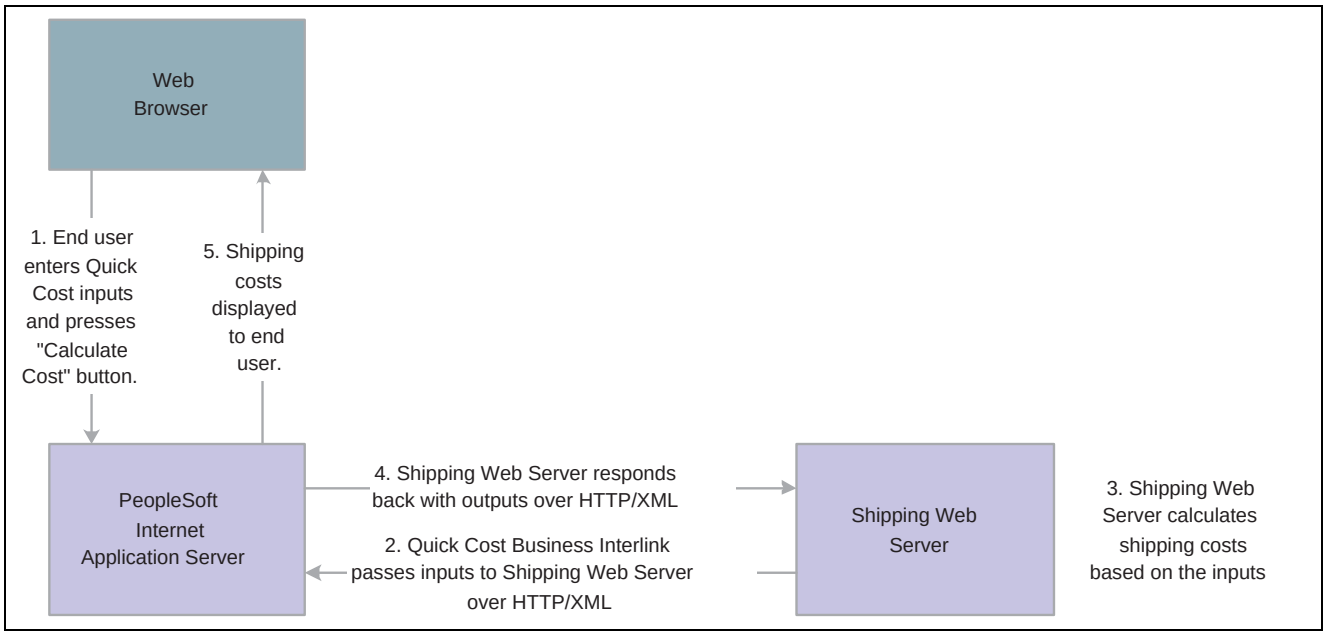
The Shipping Time & Cost example shows how Business Interlinks can be used.

You can create a Shipping Time & Cost page to trigger two different Business Interlinks that call out to the PScustomer shipping and tracking website 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 click the Calculate button. At this point, the Business Interlink is invoked on the application server, and it issues an HTTP request out to the PScustomer website, which calculates the shipping costs and returns the information back to the PeopleSoft application.

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 transaction inputs and outputs.

Note. The Shipping Time & Cost (freight carrier) example is for demonstration purposes only and is not delivered.

Example of Creating a Calculate Cost Business Interlink

This section discusses:

- A design-time plug-in example for a Business Interlink that calculates shipping costs.
- A runtime plug-in example for a Business Interlink that calculates shipping costs.
- A Business Interlink definition example for a Business Interlink that calculates shipping costs.
- A PeopleCode example for a Business Interlink that calculates shipping costs.

Design-Time Plug-in Example

The design-time plug-in is written in XML, and defines the shape of the Business Interlink. The shape is the inputs and outputs for the transaction. Using XML to create a design-time plug-in allows you to easily organize the inputs and outputs into a hierarchical structure.

Following is a simplified XML example for a transaction named Calculate Cost.

```
<transaction name="Calculate Cost">
   <input_list>
      <input name="From">
      <input name="To">
      <input name="Package_Info">
   </input_list>
   <output_list>
      <output name="Service_Rate">
   </output_list>
</transaction>
```

Runtime Plug-in Example

The runtime plug-in can be written in C++, Visual Basic, or Java. It uses the Business Interlink object as its input and output. The Business Interlink object contains the inputs, outputs, and Business Interlink methods. Since the runtime plug-in encapsulates the external system, PeopleSoft applications can treat all Business Interlinks as a PeopleSoft object.

The runtime plug-in performs the following tasks:

- Connects to the external system.
- Passes inputs to the external system.
- Receives outputs from the external system.

A runtime plug-in must be written specifically for each external system's interface architecture.

Following is a simplified runtime plug-in pseudo-code example for a transaction named Calculate Cost. This transaction gets inputs from the Business Interlink object IntObj, passes them to a function provided by the third party that calculates the cost, or ServiceRateValue, and then receives the cost and passes it to the Business Interlink object.

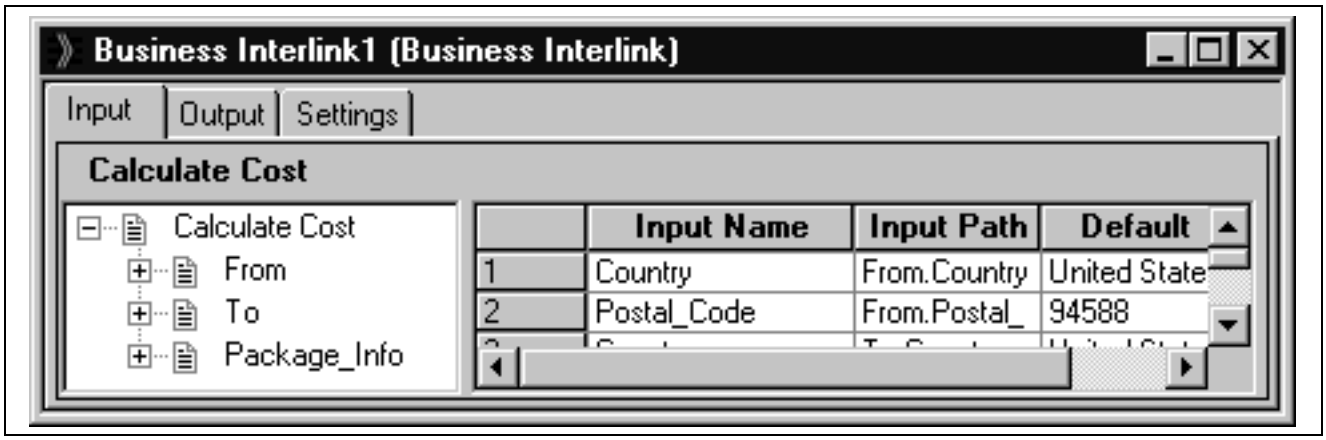
```
ExecuteTransaction(InterfaceObject IntObj)
{
   docsOut = IntObj->GetOutputDocs();
```

```
   docsIn = IntObj->GetInputDocs();

   if(IntObj->GetObjName() == "Calculate Cost") {
      FromValue = docsIn.GetValue("From");
      ToValue = docsIn.GetValue("To");
      PackageInfoValue = docsIn.GetValue("Package_Info");

      ExternalCallToCalcCost(FromValue, ToValue, PackageInfoValue,
         ServiceRateValue)

      docsOut.AddValue("Service_Rate", ServiceRateValue)
   }
}
```

Business Interlink Definition Example

The Business Interlink definition is created using PeopleSoft Application Designer, and it contains the specific shape that you want to use for your Business Interlink. Within PeopleSoft Application Designer, you select the XML design-time plug-in for your external system, then you select the transaction, and then you select the input and output parameters that you want to use with that transaction (required parameters are selected automatically).

Following is a sample of a new Business Interlink definition.



Business Interlink search dialog box

Calculate Cost PeopleCode Example

Following is a simplified PeopleCode example for a transaction named Calculate Cost. This PeopleCode gets the Business Interlink definition, BI_COST, and creates a Business Interlink object, BI_COST_OBJ. The PeopleCode program adds input values to the Interlink object. The PeopleCode program executes the Interlink object, which sends inputs to the runtime plug-in, executes the runtime plug-in, and receives outputs from the runtime plug-in. Then the PeopleCode program gets the output values from the Interlink object.

```
Local Interlink &BI_COST_OBJ;
Local BIDocs &inDoc;
Local BIDocs &outDoc;
Local boolean &RSLT;
Local float &RATE;
&BI_COST_OBJ = GetInterlink(INTERLINK.BI_COST);
```

```
/* ===> Add inputs */
&InputDocs = &BI_COST_OBJ.GetInputDocs("");
&ret = &InputDocs.AddValue("From", "San Francisco");
&ret = &InputDocs.AddValue("To", "New York");
&ret = &InputDocs.AddValue("Package_Info", "Standard");

/* ===> Execute the Interlink */
&EXECRSLT = &BI_COST_OBJ.Execute();
If ( &EXECRSLT <> 1 ) Then
   /* The instance failed to execute */
Else
   &outDoc = &BI_COST_OBJ.GetOutputDocs("");
   &ret = &outDoc.GetValue("Service_Rate", &RATE);
End-If; /* If NOT &RSLT ... */
```

CHAPTER 3

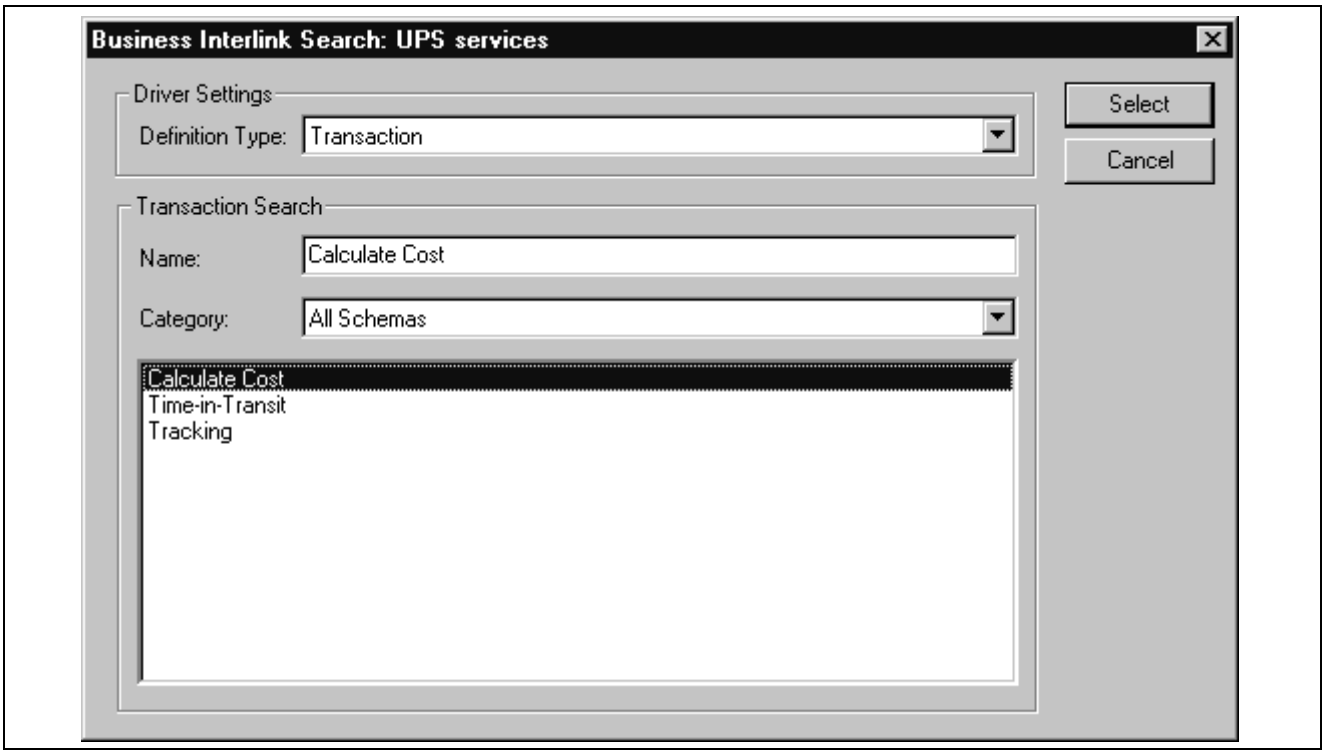
Designing a Business Interlink Definition

This section discusses how to:

- Design a Business Interlink definition, using a freight carrier plug-in as an example.
- Set inputs for a Business Interlink definition.
- Set outputs for a Business Interlink definition.
- Set configuration parameters for a Business Interlink definition.
- Test a Business Interlink definition.
- Use the Business Interlink page.

Designing a Business Interlink Definition

Use the Application Designer to design the Business Interlink definition. You need to know how to access the PeopleSoft Application Designer to use these instructions.



Business Interlink search dialog box

To design a Business Interlink definition

1. Start the PeopleSoft Application Designer.
2. Select File, New. Then from the New dialog box, select Business Interlink and click OK.
3. From the New Business Interlink dialog box, select the name of the desired Business Interlink plug-in and click OK.
4. From the Business Interlink Search dialog box, select the transaction from which you want to build the Business Interlink definition.

   For transactions, select Transaction from the Definition Type drop-down list. Then either type the name of the transaction into the Name field and click Select, or fill the Transaction Search list box with transaction names and then double-click on the desired name.

To fill the Transaction Search list box or Class Search list box:

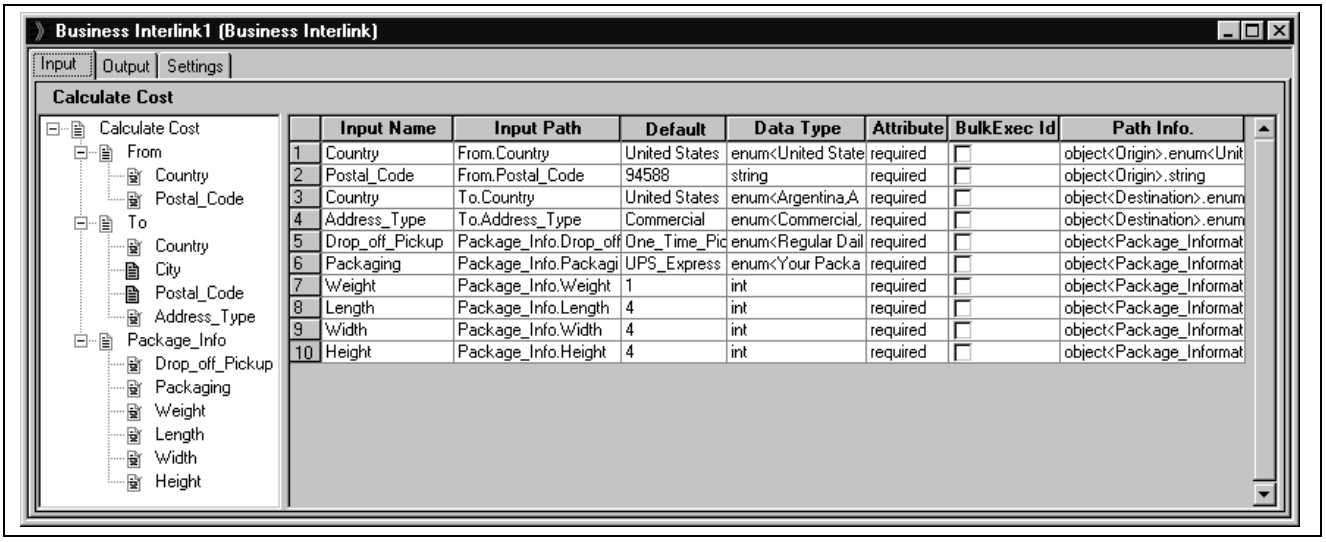
1. Click Select to fill it with all the transaction or class names, or
2. Select a category from the Category drop-down list to list only the transaction or class names in that category, or
3. Type the first part of the transaction or class name into the Name field and click Enter to get a list of transaction or class names that start with the typed text.
4. Set the parameters for this Business Interlink definition with the Input or Output tabs. For transactions, use the Input and Output tabs to set the input and output parameters for this Business Interlink definition.
5. If needed, set the Configuration Parameters with the Settings tab.
6. After you have set the Inputs, Outputs, and Settings as you want them, select File, Save to save your Business Interlink definition.
7. To test your Business Interlink definition, either select View, Test or click the Test button on the toolbar.

Next, you will use PeopleCode to instantiate your Business Interlink definition, create a Business Interlink object, and then run it.

Setting Business Interlink Inputs

For transactions, you click the Input tab, and then set the inputs for your Business Interlink definition.

In this case, this is the page for Calculate Cost(Domestic). Note that default values are in the Default column; these values were created in the design-time plug-in file, PScustomer.xml.



	Input Name	Input Path	Default	Data Type	Attribute	BulkExec Id	Path Info.
1	Country	From.Country	United States	enum<United State	required		object<Origin>.enum<Unit
2	Postal_Code	From.Postal_Code	94588	string	required		object<Origin>.string
3	Country	To.Country	United States	enum<Argentina,A	required		object<Destination>.enum
4	Address_Type	To.Address_Type	Commercial	enum<Commercial,	required		object<Destination>.enum
5	Drop_off_Pickup	Package_Info.Drop_off	One_Time_Pic	enum<Regular Dail	required		object<Package_Informat
6	Packaging	Package_Info.Packagi	UPS_Express	enum<Your Packa	required		object<Package_Informat
7	Weight	Package_Info.Weight	1	int	required		object<Package_Informat
8	Length	Package_Info.Length	4	int	required		object<Package_Informat
9	Width	Package_Info.Width	4	int	required		object<Package_Informat
10	Height	Package_Info.Height	4	int	required		object<Package_Informat

Business Interlink dialog box: Input tab

To set the value for an input (or override the default value for that input), type a value in the Default column. The Default column lists the default value for every input, if one exists.

You cannot delete Required inputs from the grid. By default, all required inputs are placed in the grid, which selects them for your Business Interlink definition.

To select an optional input for your Business Interlink definition, drag it from the tree control and drop it on the grid. (You cannot select an input object shown with a + or − to the left of its name.)

The Input Name column contains the display name of the input. This is the name that will be used for the input within PeopleCode. You can select and edit the input name in the Input Name column, and the edited name will be used for the input within PeopleCode.

The BulkExec ID column is used when you use the BulkExecute PeopleCode method instead of the Execute PeopleCode method to execute the Business Interlink. Check the check boxes that are key fields; these fields are the input record fields that are duplicated in the output record.

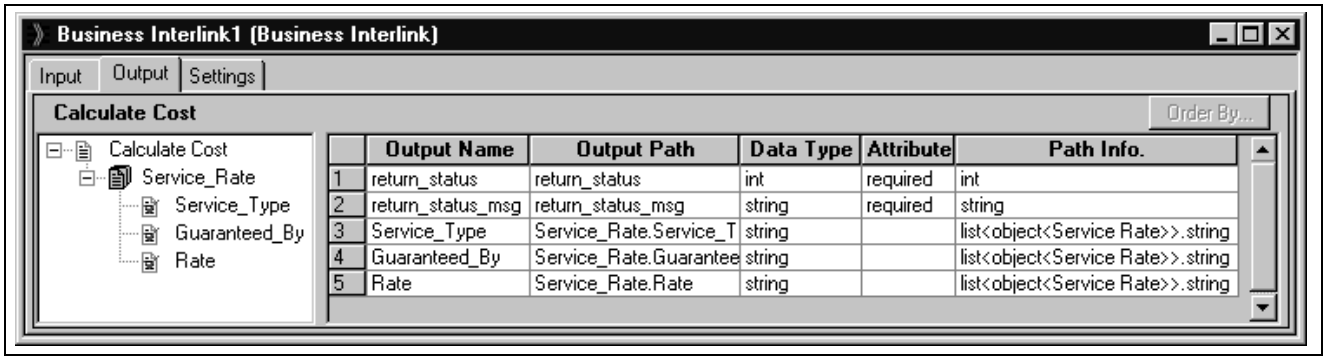
See Chapter 6, “Business Interlink Class Methods and Properties,” BulkExecute, page 44 and Chapter 5, “Running a Business Interlink Object,” page 27.

Setting Business Interlink Outputs

For transactions click the Output tab, and then set the outputs for your Business Interlink definition.

To select an output for your Business Interlink definition, drag it from the tree control and drop it on the grid.

The Output Name column contains the display name of the output. This is the name that will be used for the output within PeopleCode. You can select and edit the output name in the Output Name column, and the edited name will be used for the output within PeopleCode. You will receive values for the outputs when you fill in the PeopleCode template for the Business Interlink object created by generating this Business Interlink definition.



Business Interlink Definition: Output tab

To select an output from that class for your Business Interlink definition, drag it from the tree control and drop it on the grid. The order of the query outputs will match the order of the data members in the grid.

To move the outputs on the grid, highlight the Output Path and click the Up or Down button to move it.

Setting Business Interlink Configuration Parameters

If you are going to test your Business Interlink definition, you may need to set your configuration parameters. For example, you may need to set them to values that connect with a running server. To set the configuration parameters, click the Settings tab.

To set the configuration parameters defaults for this Business Interlink plug-in, enter the values into the Defaults column.

To set the defaults of the configuration parameters for this driver to the values you have entered in the Defaults column, enter the values and then click the Set Default button to save the values in the driver database. The next person to create a Business Interlink definition for this Business Interlink plug-in will have the Configuration Parameter defaults that you entered here. These default values are also used when you instantiate the Business Interlink definition, creating a Business Interlink object, and then run that Business Interlink object using PeopleCode. You can override these default values in PeopleCode.

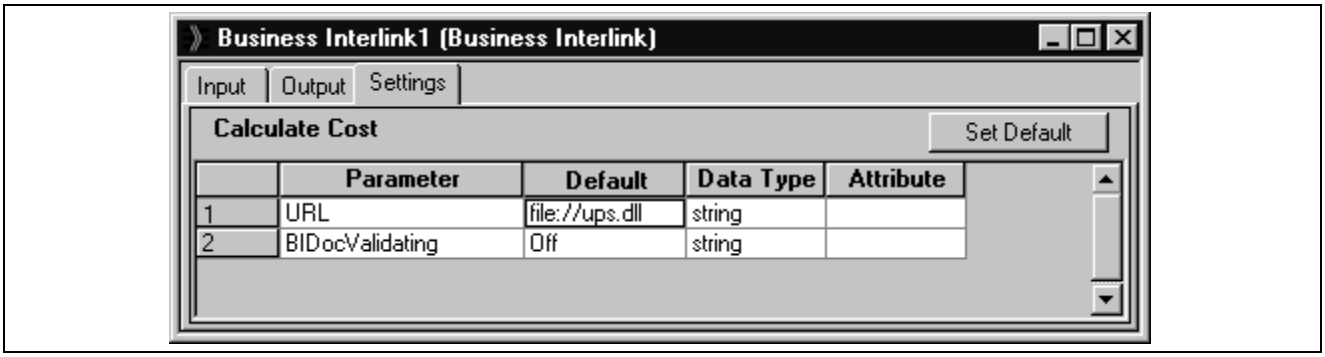
The configuration parameters for this plug-in are:

URL Set to the location of the Business Interlink runtime plug-in.

BIDocValidate When set to ON, a BIDocs object is checked to see if it exists before adding/getting values from it. For example, if the BIDoc object "Rate" does not exist, and BIDocValidate is set to ON, then the following line of code will cause an error:

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

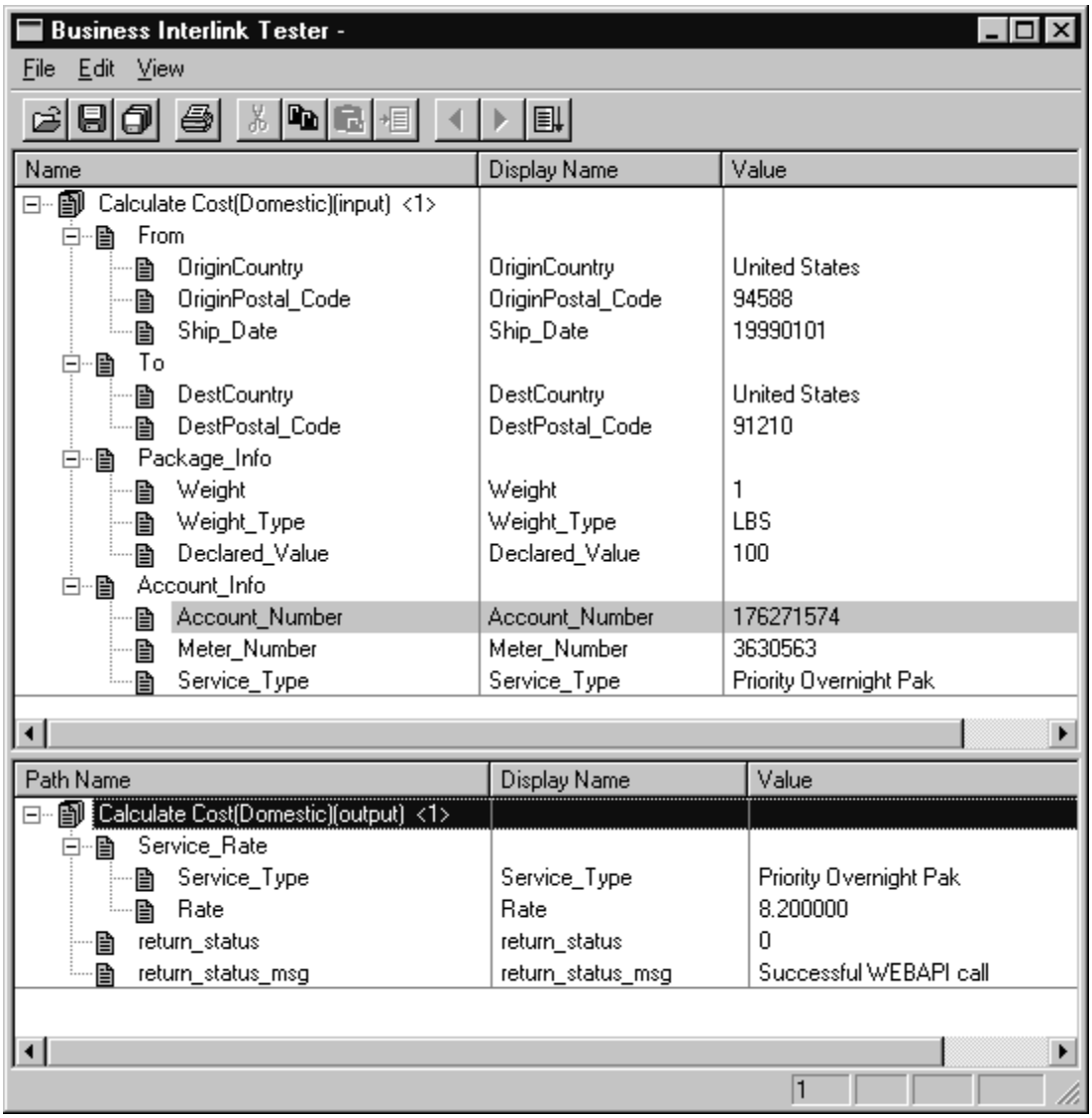
See Chapter 6, "Business Interlink Class Methods and Properties," Using BIDocs Hierarchical Methods, page 65.



Business Interlink dialog box: Settings tab

Testing Your Business Interlink Definition

The Business Interlink Tester is two-tier only. To use the Business Interlink Tester, you must have PeopleTools installed on your system.



Business Interlink Tester

To run the Business Interlink Tester

1. Connect to any external system that your Business Interlink needs.
2. Start the Business Interlink Tester.

   Open your Business Interlink (if it is not yet open) and either select View, Test or click the Test button.
3. From the Select Transaction window, select the Business Interlink transaction that you want to test.

   Once you select your transaction, the Business Interlink Tester will show all of the inputs and outputs for that transaction. This example tests the Calculate Cost(Domestic) transaction.
4. In the Business Interlink Tester page showing the inputs and outputs for your Business Interlink Transaction, enter input values for your transaction.

   If the XML design-time plug-in already defined values for them, they will appear in the value column. You can enter input values two ways:

   • Type the input values into the Value column.

- If you have saved a set of input values as an Input Doc, select Open Input Doc, navigate to the location of your Input Doc, and then select the Input Doc. Step 5 shows how to save an Input Doc.

5. If you wish to save your Input values, select File, Save Input Doc or File, Save Input Doc As.

   You can then use those input values when you run the Business Interlink Tester on this transaction at a later time.

6. You can enter more than one set of inputs, or input docs, to test, provided your Business Interlink plug-ins support it.

   - To enter more than one input doc, highlight the root level of the input grid, click the Insert new doc button, and enter input values into the new input doc.
   - To move to each input doc, use the previous doc and next doc buttons.

7. Once you have entered all the required input values, and the Test button is enabled, click the Test button to test your Business Interlink plug-ins.

   Once the test has run, the bottom grid will contain a row of data for each output that you earlier defined under the Output tab.

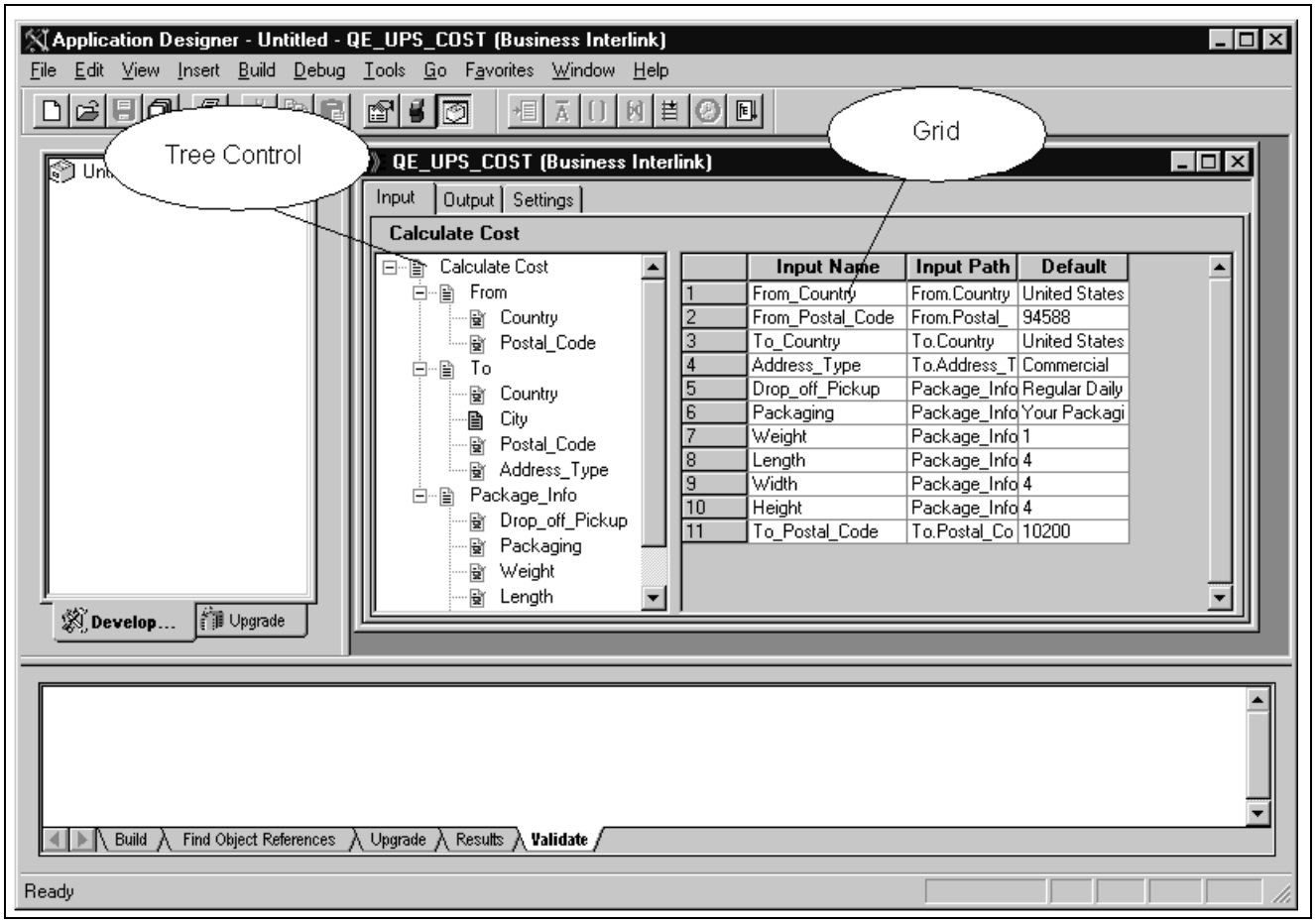
8. If you entered multiple input docs and received multiple output docs as a result, view the multiple output docs by highlighting the root level of the output grid and clicking the previous doc and next doc buttons.

9. If you want to print your inputs or outputs, highlight the inputs or outputs and then select Print from the File menu.

Understanding the Application Designer Business Interlinks

This section describes how to use the tree control and grid when creating a Business Interlink in the designer view. This view appears when you click the Input, Output, or Settings tab, which you use to set the parameters for your transaction Business Interlink definition. The Business Interlink page has a tree control on the left side of the page, and a grid on the right side. The tree control lists either transaction input/output parameters, class data members, or configuration parameters. You will select them for your Business Interlink definition by performing any of the following actions:

- To expand the transaction, class, or object instances to show its inputs/outputs/data members, click the + to the left of the transaction/class/object instance.
- To select an input/output/data member for your Business Interlink definition, drag it from the tree control and drop it on the grid. If you select an object (shown by a + or − to the left of its name), you select that object's OID for your Business Interlink definition.
- Each row in the grid contains an input, output, or data member. To remove a row from the grid, highlight it by clicking on the corresponding number in the leftmost column of the grid, and either click Delete or select File, Delete.
- To add an empty row to the grid, highlight a row and click the Add Row button.
- To display the Data Type, Attribute (required status), and path information of an input/output/data member/configuration parameter, and the BulkExec ID for an input parameter, either right-click in the grid and select More Info from the menu, or click the Datatype button.
- To run the Business Interlink Tester after you have designed your Business Interlink, either select View, Menu or click the test button.



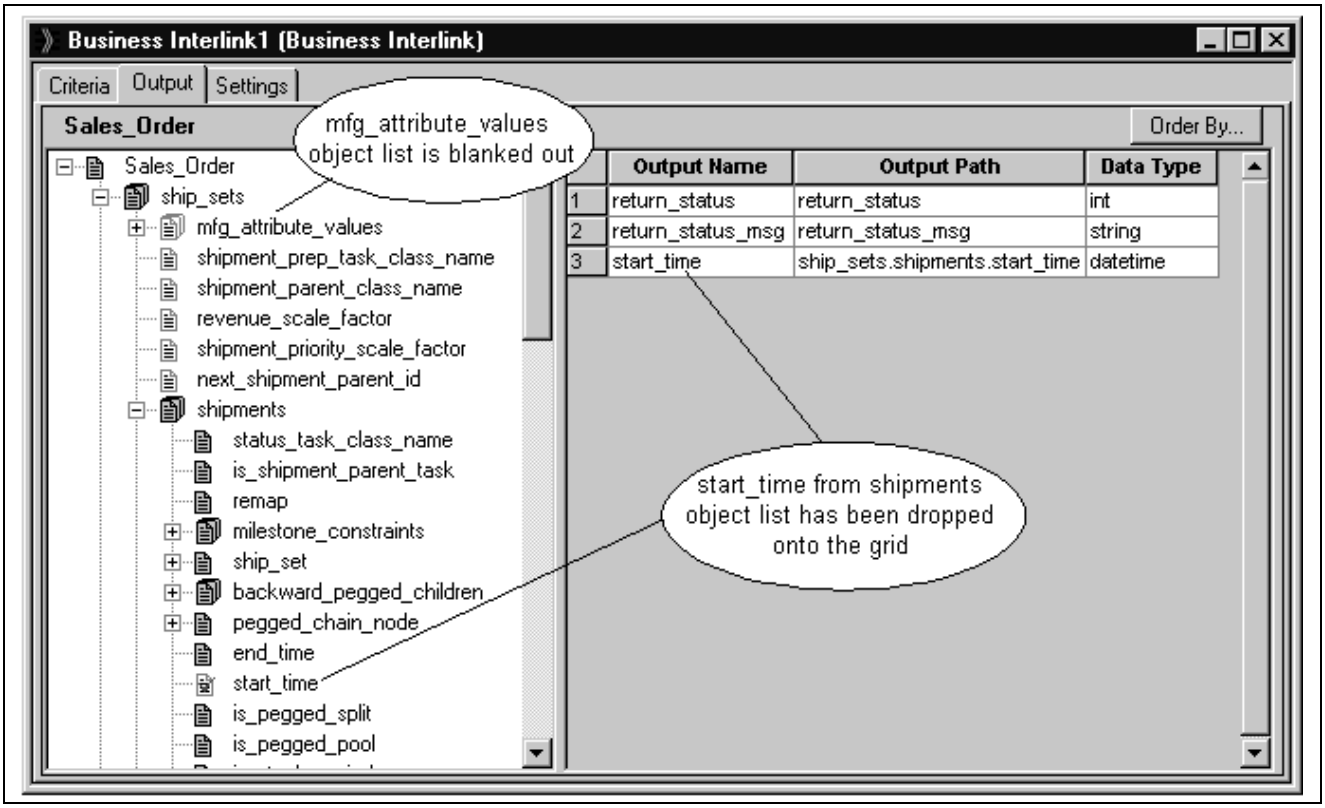
Business Interlink definition view showing the tree control and the grid

Understanding Restrictions on the Control Tree

There are restrictions on which data members you can drag and drop from the control tree to the grid. These restrictions are based on the data members that you have selected that are also lists. Lists are represented in the tree control by a multiple branch icon, such as is shown in the following figure. If a data member is made unavailable by your previous selections, you will not be able to drag it from the tree control to the grid.

This can happen if you open a list of objects in the control tree, and then drag a data member from that object list to the grid; the other object lists in the control tree will become unavailable. You will not be able to open another object list at the same level or at higher levels in the control tree and drag any of its data members to the grid.

For example, where mfg_attribute_values and shipments are both object lists, you can drag a data member of shipments to the grid. Subsequently, mfg_attribute_values would become unavailable, and you cannot open it or drag any of its data members to the grid. This is shown below, where dragging and dropping the data member start_time from the shipments object list caused the mfg_attribute_values object list to become unavailable.



Control Tree with an unavailable object list

Another restriction exists at runtime. This restriction does not cause lists in the control tree to become unavailable, but it can cause runtime errors: if you drag and drop a list to the grid, any other list that you drag and drop must have the same number of entries as the previously dropped list. At runtime, the Business Interlink plug-in determines the number of entries in the list.

CHAPTER 4

Generating the PeopleCode Template

This chapter provides an overview of how to generate PeopleCode templates and discusses how to:

- Generate a PeopleCode template in a PeopleCode event.
- Generate a PeopleCode template in an Application Engine program.

Understanding How to Generate a PeopleCode Template

This section shows how to generate a Business Interlink PeopleCode template, using the freight carrier plug-in as an example.

Note. The freight carrier example is for demonstration purposes only and is not delivered.

You will use PeopleCode to run a Business Interlink object, which is an instantiation of a Business Interlink definition. You need to know how to access the PeopleSoft Application Designer to use these instructions.

You can run a Business Interlink object as part of an online process (PeopleCode Event on a record, component, page, and so on) or a batch process (that is, an Application Engine program.) The example shows creating a program in the FieldChange event for a record field, but you could run it from any event or even as part of a message subscription.

Generating a PeopleCode Template in an Event

This section discusses how to generate a Business Interlink PeopleCode template in a PeopleCode event, using the PScustomer plug-in as an example.

To generate a Business Interlink PeopleCode template in a FieldChange event:

1. Start the PeopleSoft Application Designer.
2. Select the PeopleSoft object from which you will add a PeopleCode event to instantiate your Business Interlink definition and execute the Business Interlink object created by the instantiation.
3. Choose the PeopleCode event from which you will instantiate and execute your Business Interlink object.

   Select View, PeopleCode Display or click the PeopleCode Display button to display the PeopleCode events.

   From this page, decide which Record Field you want to enter your Business Interlink object, then select the PeopleCode Event into which you want to instantiate and execute your Business Interlink object by double-clicking on that Record Field's PeopleCode event.

Once you double-click the event, the PeopleCode editor window for that Record Field's PeopleCode event appears.

The PeopleCode editor appears after you double-click the FieldChange check box.

PeopleCode is associated with many components and many events. Usually, the object is a Record Field, and the event is an action that an end-user takes upon a PeopleSoft page that references that object. Such an event could be changing data in a Record Field displayed on the page (Field Change). It could also be saving the information on that page. The PeopleCode is executed when that event occurs.

4. Insert your Business Interlink definition into the Project tree control on the left side of the Application Designer page.

   There are several ways to do this. One way is to select File, Open. Then from the Open Object dialog box, select *Business Interlink* from the Object Type drop-down list box, click the Open button to get a list of Business Interlinks in the Objects matching selection criteria list, then double-click on the Business Interlink that you want to use. Then select Insert, Current Object into Project.

   Another way is to select Insert, Objects into Project. Then from the Insert into Project dialog box, select *Business Interlink* from the Object Type drop-down list box, click the Insert button to get a list of Business Interlinks in the Objects matching selection criteria list, then double-click on the Business Interlink that you want to use.

   If you have already inserted your Business Interlink into a project, you can select File, Open, select Project, click the Open button to list the projects, and select your project that contains this Business Interlink. That project, along with your Business Interlink, will then be displayed in the Project control tree.

5. Generate your PeopleCode template.

   Generate a PeopleCode template in order to avoid a lot of typing. For example, the template contains calls to the Business Interlink object methods, and includes the names of the inputs and outputs for your Business Interlink object.

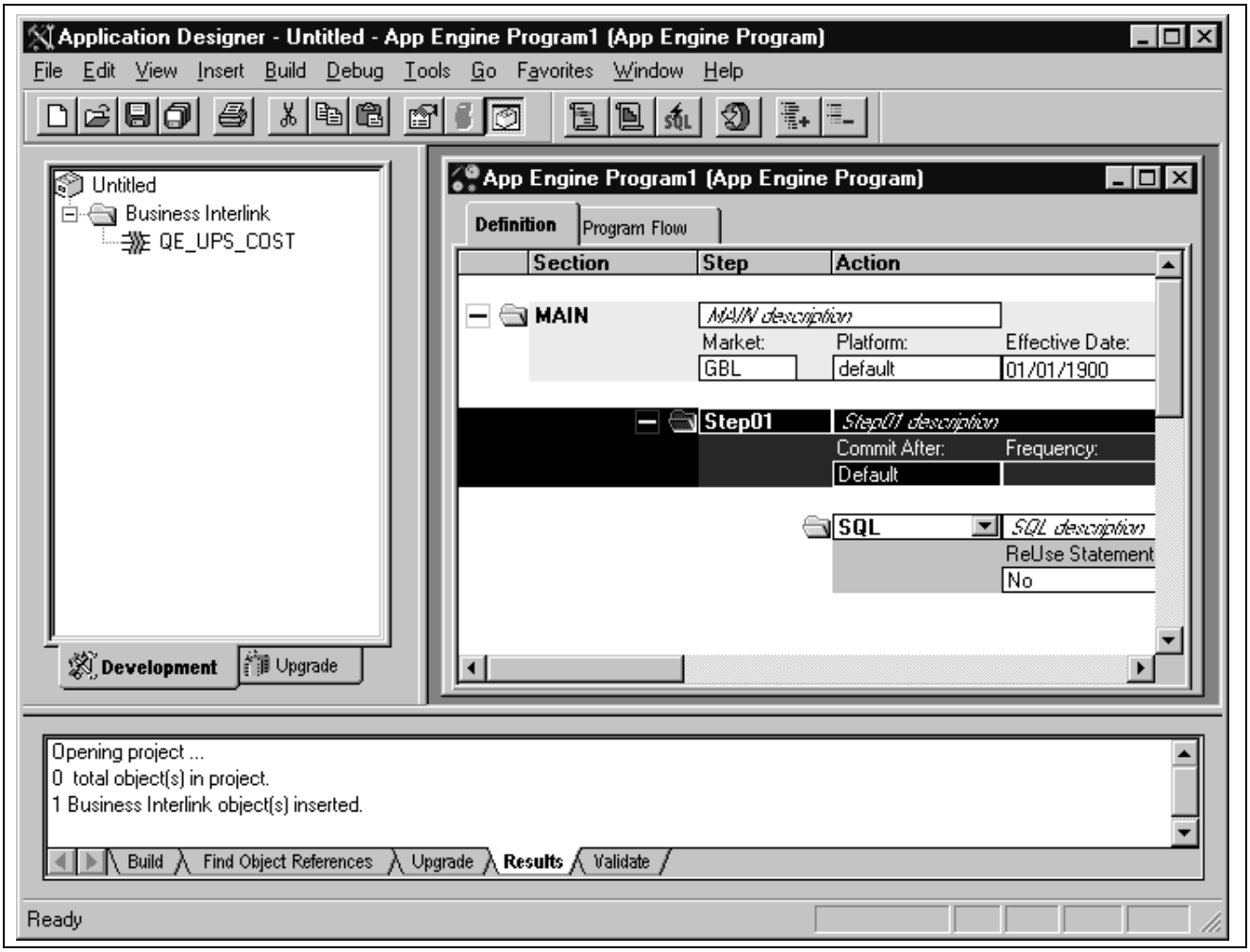
   Generate the template by dragging and dropping your Business Interlink name from the Project tree control on the left side into the open PeopleCode editor window on the right side. A PeopleCode template is generated for your Business Interlink within the PeopleCode editor window.

6. Fill in the PeopleCode template so that this PeopleCode program can run your Business Interlink object.

See Chapter 5, "Running a Business Interlink Object," page 27 and *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Developer's Guide*, "PeopleCode and the Component Processor".

Generating a PeopleCode Template in an Application Engine Program

This section discusses how to generate a Business Interlink PeopleCode template in a PeopleCode event, using the PScustomer plug-in as an example.



Application Engine program: action added

To generate a Business Interlink PeopleCode template for an Application Engine program:

1. Start the PeopleSoft Application Designer.
2. Insert your Business Interlink definition into the Project tree control on the left side of the Application Designer page.

   For example, you can select File, Open to select a Project from the dialog box that contains the Business Interlink that you want to use with the Application Engine Program, then insert the Business Interlink. In the following step, Insert, Objects into Project is used to insert the Business Interlink.
3. Open or create an Application Engine Program.

   For example, you can select File, New, and then select *App Engine Program* from the dialog box.

   When you have opened the Application Engine Program, you get a screen similar to the following:
4. If this Application Engine Program does not have an action, or you want to create a new action for the PeopleCode program, select Insert, Action. In this case, you must highlight a step (Step01) before you can insert an action.

   You will see the new action added.
5. In the action, select PeopleCode from the drop-down menu.

6. If you have not done so, save the Application Engine Program. Then right-click on the PeopleCode action, and then select View PeopleCode. An empty PeopleCode editor window appears.
7. Generate your PeopleCode template.

   Generate a PeopleCode template in order to avoid a lot of typing. For example, the template contains calls to the Business Interlink object methods, and includes the names of the inputs and outputs for your Business Interlink object.

   Generate the template by dragging and dropping your Business Interlink name from the Project tree control on the left side into the open PeopleCode editor window on the right side. A PeopleCode template is generated for your Business Interlink within the PeopleCode editor window.
8. Fill in the PeopleCode template so that this PeopleCode program can run your Business Interlink object.

CHAPTER 5

Running a Business Interlink Object

This chapter discusses how to:

- Write a PeopleCode program to execute a Business Interlink object.
- View an example of a transaction PeopleCode template.
- View an example of a fully-coded transaction PeopleCode template.
- Name inputs and outputs with limitations.

Writing the PeopleCode Program to Execute a Business Interlink Object

This section shows how to run a Business Interlink object using PeopleCode.

See *Enterprise PeopleTools 8.45 PeopleBook: PeopleCode Developer's Guide*.

To run the Business Interlink object, you write a PeopleCode program that executes the Business Interlink object. The PeopleCode template contains the code that you can start with after you drag a Business Interlink definition into a PeopleCode page. The template generates the main structure of PeopleCode to instantiate your Business Interlink definition, creating a Business Interlink object, and then adding input values to, executing, and fetching output values from that Business Interlink object.

When you write a PeopleCode program, you fill in the PeopleCode template. (You could write the PeopleCode program from scratch, but it is easier to start with a template.) Filling in the PeopleCode template means writing a PeopleCode program that instantiates your Business Interlink definition into a Business Interlink object, and then executes the Business Interlink object. Once this has been done, the Business Interlink object is run whenever the PeopleCode program is executed. The PeopleCode program can be executed from an event or from an Application Engine Program, depending upon which you chose when you generated the PeopleCode template.

See Chapter 4, "Generating the PeopleCode Template," Understanding How to Generate a PeopleCode Template, page 23.

To fill in the PeopleCode template, you can replace all references to <*> with references to PeopleCode variables or to Record Fields, and by inserting values for the input parameters:

- For Business Interlink object inputs, within the AddInputRow call, replace any references to <*> with a constant value, or a Record Field name, or a PeopleCode variable name.
- For Business Interlink object outputs, within the FetchNextRow call, replace any references to <*> with a Record Field name or a PeopleCode variable name.
- There may be default values for the input parameters in the PeopleCode template. Optionally, you can replace default values for the input parameters by typing in new constants, Record Fields, or PeopleCode variables.

The PeopleCode template consists of the following main sections:

- Configuration Parameter settings. If they have default values, you can type new values for them if you desire. Replace any <*> with a constant value, Record Field, or PeopleCode variable.
- **GetInterlink** method. This method instantiates your Business Interlink definition, creating a Business Interlink object that you can execute within PeopleCode.
- **AddDoc, AddValue** method. This section passes one set of input parameter values to the Business Interlink object. You can use several methods to pass inputs:
  - The supplied **GetInputDocs**, **AddDoc**, and **AddValue** methods pass a row of input to the object. Replace each <*> with a constant value, Record Field, or PeopleCode variable; these are the input values that you pass to the transaction. If the inputs have default values, you can type new values for them if you desire. The input is stored in an input buffer containing one or more sets of input parameter values for this transaction. If you pass only one set of input parameter values, the transaction is executed in real time. If you place the **AddDoc** and **AddValue** methods in a loop (including the **AddNextDoc** method to add the next set of input parameters) and call them many times to add many sets of input parameter values to the input buffer, the transaction is executed in batch mode. Queries have one input row, and are executed in real time.
  - If your data is mapped to a rowset, you can replace the **GetInputDocs**, **AddDoc**, and **AddValue** methods with the **InputRowset** method. The **InputRowset** method will take a standard rowset object to populate the input buffers.
  - If you're sending a large amount of data to the input buffers, you might delete the supplied the **GetInputDocs**, **AddDoc**, and **AddValue** methods and write the data to a PeopleSoft record, then use the **BulkExecute** method. The **BulkExecute** method uses the data in the record to populate the input buffer, copying like-named fields. This method assumes that the names of the fields in the record match the names of the inputs defined in the Business Interlink definition.
- **Execute** method. This section executes the Business Interlink object. You can use several methods to execute:
  - The supplied **Execute** method executes the transaction. You call this method once, and it will execute once for each row of input you have in the input buffer. (Since queries do not have multiple rows of input, they are executed once.)
  - If you're sending a large amount of data to the input buffers and you wrote the input data to a staging table, replace the **Execute** method with the **BulkExecute** method.
- **GetOutputDocs, GetDoc,** and **GetValue** methods. This section gets the outputs from the Business Interlink object. You can use several methods to get the outputs:
  - The supplied **GetOutputDocs**, **GetDoc**, and **GetValue** methods get one set of output parameter values. For a transaction that was executed in batch mode (more than one set of input parameter values entered), you will usually get one set of output for each set of input. Place the **GetDoc** and **GetValue** methods into a loop to do so, and include the **GetNextDoc** method to get the next set of output parameters. Replace each <*> with a Record Field or PeopleCode variable; these will be where the output values from the transaction is stored.
  - If you used the **BulkExecute** method, it can automatically fill the output record specified with the method with the output values if you've specified an output record.
  - If your data is hierarchical and you used the **InputRowset** method to take a standard rowset object to populate the input buffers, then replace the **GetOutputDocs**, **GetDoc**, and **GetValue** methods with the **FetchIntoRowset** method. The **FetchIntoRowset** method populates the rowset with data.

See Chapter 6, "Business Interlink Class Methods and Properties," page 37.

Viewing an Example of a Transaction PeopleCode Template

When you drag over the Business Interlink definition QE_COST, which is for the transaction Calculate Cost, you get the following PeopleCode. This Business Interlink definition is supplied with the Business Interlink software.

In addition to the template code and comments, this example includes some additional comments, marked with the text *ADDITIONAL COMMENT.*

```
/* ===>
This is a dynamically generated PeopleCode template to be used only as a helper?
 to the application developer. You need to replace all references to '<*>' OR default?
 values with references to PeopleCode variables and/or a Rec.Fields.*/



/* ===> Declare and instantiate: */
Local Interlink &QE_COST_1;
Local BIDocs &inDoc;
Local BIDocs &outDoc;
Local boolean &RSLT;
Local number  &EXECRSLT;
&QE_COST_1 =  GetInterlink(INTERLINK.QE_COST);

/* ===> You can use the following assignments to set the configuration parameters.
*/

/* ADDITIONAL COMMENT: The URL parameter points to the Business Interlink?
 runtime plug-in.*/
&QE_COST_1.URL = file://PScustomer.dll;


/* ===> You might want to call the following statement in a loop if you have?
 more than one row of data to be added. */

/* ADDITIONAL COMMENT: The AddValue calls use for its inputs the input names?
 in the Input Path column of the Input Tab for this Business Interlink Definition.

The AddValue calls also contain all of the default values that have been set?
 for the QE_COST Business Interlink Definition. You will likely replace?
 them with variables or record fields when you complete the coding for this?
 template. */

/* ===> Add inputs: */
&inDoc = &QE_COST_1.GetInputDocs("");
&FromDoc = &inDoc.AddDoc("From");
&ret = &FromDoc.AddValue("Country", <*>);
&ret = &FromDoc.AddValue("Postal_Code", <*>);
&ToDoc = &inDoc.AddDoc("To");
&ret = &ToDoc.AddValue("Country", <*>);
```

```
&ret = &ToDoc.AddValue("Address_Type", <*>);
&ret = &ToDoc.AddValue("Postal_Code", <*>);
&Package_InfoDoc = &inDoc.AddDoc("Package_Info");
&ret = &Package_InfoDoc.AddValue("Drop_off_Pickup", <*>);
&ret = &Package_InfoDoc.AddValue("Packaging", <*>);
&ret = &Package_InfoDoc.AddValue("Weight", <*>);
&ret = &Package_InfoDoc.AddValue("Length", <*>);
&ret = &Package_InfoDoc.AddValue("Width", <*>);
&ret = &Package_InfoDoc.AddValue("Height", <*>);


/* ===> The following statement executes this instance: */
&EXECRSLT = &QE_COST_1.Execute();
If ( &EXECRSLT <> 1 ) Then
   /* The instance failed to execute */
Else
/* ADDITIONAL COMMENT: The GetValue calls use for the outputs the output names?
 in the Output Path column of the Output Tab for this Business Interlink?
 Definition. */
&outDoc = &QE_COST_1.GetOutputDocs("");
&Service_RateDoc = &outDoc.GetDoc("Service_Rate");
&ret = &Service_RateDoc.GetValue("Service_Type", <*>);
&ret = &Service_RateDoc.GetValue("Guaranteed_By", <*>);
&ret = &Service_RateDoc.GetValue("Rate", <*>);
&ret = &outDoc.GetValue("return_status", <*>);
&ret = &outDoc.GetValue("return_status_msg", <*>);

End-If; /* If NOT &RSLT ... */
```

Viewing an Example of a Fully-Coded Transaction PeopleCode Template

The following code is an example of how the template for the Business Interlink definition QE_COST could be coded. This PeopleCode is an edited version of the PeopleCode contained within the QE_COST record, QE_COST_BUTTON field, FieldChange PeopleCode event.

This code executes the transaction named Calculate Cost(Domestic), which calculates the cost of shipping a package. It also execute the Tracking transaction, which tracks a package.

Following this code is an example of the inputs and outputs for these transactions, and then a short discussion of the records where this PeopleCode is used.

In addition to the code and comments in this PeopleCode event, this example includes comments pointing out the changes made to the template. These comments are marked with the text *CHANGE*.

```
/* ===> Declare and instantiate: */
Local Interlink &QE_COST_1;
Local boolean &RSLT;
Local number &EXECRSLT;
```

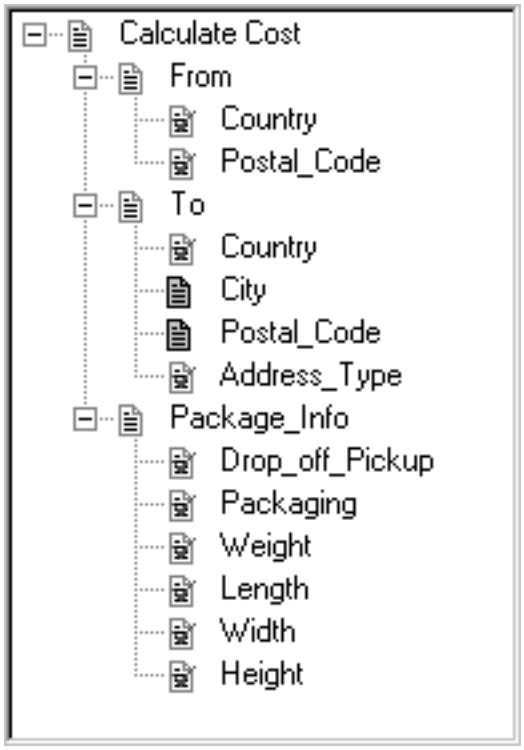
```
Local Record &REC;
Local number &count;
Local BIDocs &biDocs, &InputDocs;
Local BIDocs &ServiceRateDoc, &FromDoc, &ToDoc, &PackageDoc;
Local number &ret;

&QE_COST_1 = GetInterlink(Interlink.QE_COST);

/* ===> You can use the following assignments to set the configuration parameters.
*/

&QE_COST_1.URL = "file://PScustomer.dll";

GetLevel0()(1).GetRowset(Scroll.QE_COST_RES).Flush();

/* ===> You might want to call the following statement in a loop if you
have more than one row of data to be added. */

/* ===> Add inputs: */
/* CHANGE: The input values are set to fields within the QE_COST record,?
 rather than to default values. */

&REC = GetLevel0().GetRow(1).GetRecord(Record.QE_COST);
&COUNTRY_FROM = &REC.QE_ORIGIN_CNTRY.Value;
&FROM_POST_CODE = &REC.QE_FROM_POST_CODE.Value;
&COUNTRY_DEST = &REC.COUNTRY_DEST.Value;
&TO_POSTAL_CODE = &REC.QE_TO_POST_CODE.Value;
&DROP_OFF_PICKUP = &REC.QE_DROP_PICKUP.Value;
&PACKAGING = &REC.QE_PACKAGING.Value;
&WEIGHT = &REC.QE_WEIGHT.Value;
&LENGTH = &REC.QE_LENGTH.Value;
&WIDTH = &REC.QE_WIDTH.Value;
&HEIGHT = &REC.QE_HEIGHT.Value;
/* &DESTINATION = &REC.QE_TO_ZIP.Value; */

/* CHANGE: Evaluate several of the parameters */
/* Fix up the County From data */
Evaluate &COUNTRY_FROM
When = "USA"
   &COUNTRY_FROM = "United States"
End-Evaluate;

/* Fix up the County dest data */
Evaluate &COUNTRY_DEST
When = "USA"
   &COUNTRY_DEST = "United States"
When = "UK"
   &COUNTRY_DEST = "United Kingdom"
When = "CAN"
   &COUNTRY_DEST = "Canada"
```

```
End-Evaluate;

/* Evaluate the translates */
Evaluate &DROP_OFF_PICKUP
When = "A"
   &DROP_OFF_PICKUP = "On Call Air";
When = "L"
   &DROP_OFF_PICKUP = "Letter Center";
When = "O"
   &DROP_OFF_PICKUP = "One Time Pickup";
When = "R"
   &DROP_OFF_PICKUP = "Regular Daily Pickup";
End-Evaluate;

Evaluate &PACKAGING
When = "E"
   &PACKAGING = "Express Box";
When = "L"
   &PACKAGING = "Letter Envelope";
When = "T"
   &PACKAGING = "Tube";
When = "Y"
   &PACKAGING = "Your Packaging";
End-Evaluate;


/* New Hierarchical data buffer stuff */

/* CHANGE: The <*> in the AddValue methods are replaced with proper values. */
&InputDocs = &QE_COST_1.GetInputDocs("");
&FromDoc = &InputDocs.AddDoc("From");
&ret = &FromDoc.AddValue("Country", "United States");
&ret = &FromDoc.AddValue("Postal_Code", &FROM_POST_CODE);

&ToDoc = &InputDocs.AddDoc("To");
&ret = &ToDoc.AddValue("Country", &COUNTRY_DEST);
&ret = &ToDoc.AddValue("Postal_Code", &TO_POSTAL_CODE);
&ret = &ToDoc.AddValue("Address_Type", "Commercial");

&PackageDoc = &InputDocs.AddDoc("Package_Info");
&ret = &PackageDoc.AddValue("Drop_off_Pickup", &DROP_OFF_PICKUP);
&ret = &PackageDoc.AddValue("Packaging", &PACKAGING);
&ret = &PackageDoc.AddValue("Weight", &WEIGHT);
&ret = &PackageDoc.AddValue("Length", &LENGTH);
&ret = &PackageDoc.AddValue("Width", &WIDTH);
&ret = &PackageDoc.AddValue("Height", &HEIGHT);

/* ===> The following statement executes this instance: */
&EXECRSLT = &QE_COST_1.Execute();
If (&EXECRSLT <> 1) Then
```

```
   /* The instance failed to execute */
Else

   /* ===> Fetch Outputs: */
/* CHANGE: The <*> in the AddValue methods are replaced with proper values. */
   &RSLT = True;
   &ROWSET = GetRowset(Record.QE_COST_RES);
   &I = 1;
   &biDocs = &QE_COST_1.GetOutputDocs("");
/* CHANGE: Because Service_Rate is a list, you must get its values using a loop.?
 GetCount gets the number of Service_Rate output parameters. */
/* Possible fix: the following line may need to be added here:
   &ServiceRateDoc = &biDocs.GetDoc("Service_Rate"); */

/* Possible fix: the following line may need to be commented out */
  &count = &biDocs.GetCount("Service_Rate");
   While (&I < &count)
      &ServiceRateDoc = &biDocs.GetDoc("Service_Rate");
      &ret = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
      &ret = &ServiceRateDoc.GetValue("Guaranteed_By", &GUAR_BY);
      &ret = &ServiceRateDoc.GetValue("Rate", &RATE);
      If &ret = 0 Then
         If &I > 1 Then
            &ROWSET.InsertRow(&I - 1);
         End-If;
         UpdateValue(QE_COST_RES.QE_SVC_TYPE, &I, &SERVICE_TYPE);
         UpdateValue(QE_COST_RES.QE_GUAR_BY, &I, &GUAR_BY);
         UpdateValue(QE_COST_RES.QE_RATE, &I, &RATE);
         &I = &I + 1;
/* Possible fix: the following line may need to be added here:
         &ret = &ServiceRateDoc.GetNextDoc(); */
      End-If;
   End-While;
End-If; /* If NOT &RSLT ... */
```

Examples of the Inputs and Outputs for the Transactions

The following example shows the structure of the inputs for the Calculate Cost transaction.



Inputs for the Calculate Cost(Domestic) transaction

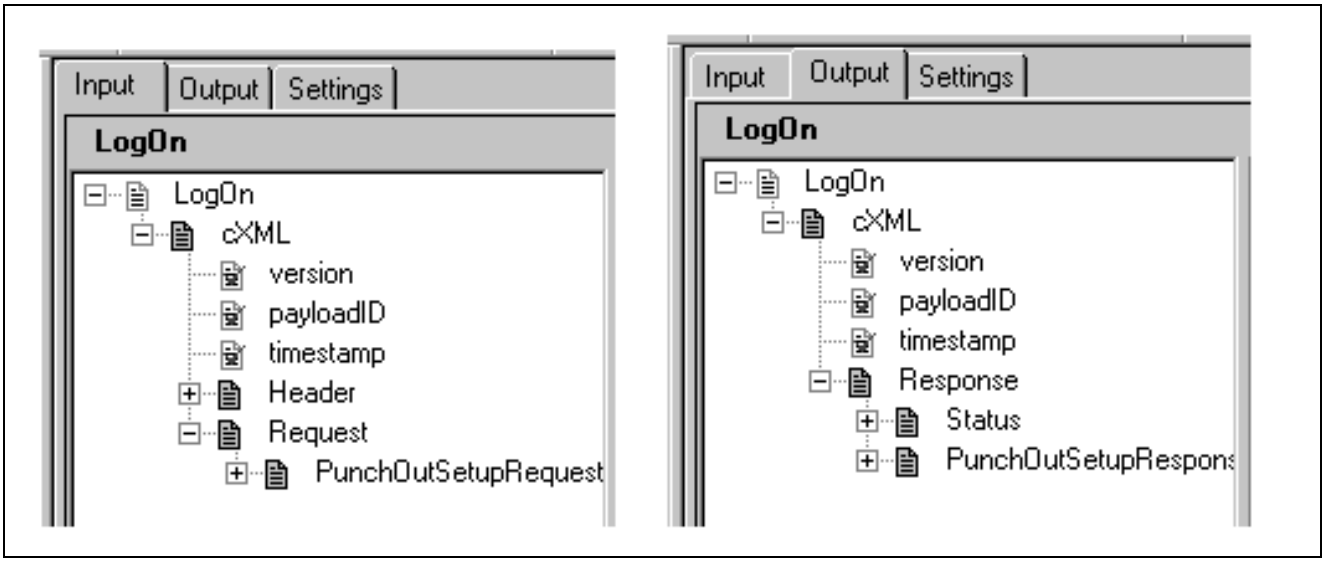
The following example shows the structure of the outputs for the Calculate Cost transaction.

Outputs for the Calculate Cost(Domestic) transaction

Naming Inputs and Outputs With Limitations

Within the PeopleCode program using Business Interlinks, the input and output for a BIDocs object should not have the same name. This situation can arise when using the BI PeopleCode auto generation feature, or if you name the input and output BIDocs with the same name. If the BI definition has a BIDoc with the same name in both the Inputs and Outputs sections, when you drag and drop to generate the PeopleCode, the code template that is generated will have BIDocs with the same name in both the Input and Output sections.

In the following example, the following Business Interlink definition, LogOn, has an input and output with the same name: cXML.



Business Interlink Definition: inputs and outputs with the same name

When you drag and drop this Business Interlink definition to create a PeopleCode template, you get the following code:

```
/* ===> Add inputs: */
&inDoc = &QE_PSQA_IMPRE_1.GetInputDocs("");
&cXMLDoc = &inDoc.AddDoc("cXML");
&ret = &cXMLDoc.AddValue("version", <*>);

&outDoc = &QE_PSQA_IMPRE_1.GetOutputDocs("");
&cXMLDoc = &outDoc.GetDoc("cXML");
&ret = &cXMLDoc.GetValue("version", <*>);
```

This code contains docs with the same name in the input and output sections. This PeopleCode will not run properly.

To fix the problem, you can rename the input doc or the output doc so that they have different names. For example:

```
/* ===> Add inputs: */
&inDoc = &QE_PSQA_IMPRE_1.GetInputDocs("");
&cXMLInDoc = &inDoc.AddDoc("cXML");
&ret = &cXMLInDoc.AddValue("version", <*>);

&outDoc = &QE_PSQA_IMPRE_1.GetOutputDocs("");
```

```
&cXMLOutDoc = &outDoc.GetDoc("cXML");
&ret = &cXMLOutDoc.GetValue("version", <*>);
```

CHAPTER 6

Business Interlink Class Methods and Properties

This chapter provides an overview of the Business Interlink class methods and properties and discusses how to:

- Decide which methods to use.
- Use the state of a Business Interlink object.
- Declare a Business Interlink object.
- Scope a Business Interlink object.
- Use Business Interlink object methods.
- Use BIDocs hierarchical methods.
- Use the Interlink property.
- Use configuration parameters.
- Use the Business Interlink function.

Understanding the Business Interlink Object Methods

After you instantiate a Business Interlink object, you'll use the Business Interlink object methods to:

- Add input values to the Business Interlink object.
- Execute the Business Interlink object.
- Get the output values from the Business Interlink object.

After the Business Interlink object is instantiated, you can assign values from constants, PeopleSoft variables, or record fields to the inputs of that Business Interlink object.

When you execute the Business Interlink object, it loads the appropriate Business Interlink plug-in and passes itself to that Business Interlink plug-in. The Business Interlink plug-in processes the input data, passing the input values of the Business Interlink object to the external system and then fills the output values of the Business Interlink object (if there are outputs).

Deciding Which Methods To Use

This section discusses how to:

- Instantiate a Business Interlink object.
- Execute a Business Interlink object.

- Support standard input and output with the BIDocs methods.
- Support batch input and output.
- Support rowsets.
- Use flat table input/output methods.

Instantiating a Business Interlink Object

You'll need to write a PeopleCode program to instantiate the Business Interlink object from the Business Interlink definition, using the **GetInterlink** function.

Executing the Business Interlink Object

In most cases, you must use the **Execute** method to execute the Business Interlink object. However, for bulk input, you can use the **BulkExecute** method instead.

The **Execute** and **BulkExecute** methods return a value you can use for status and error checking.

Supporting Standard Input/Output with BIDocs Methods

A Business Interlink object can use hierarchical input and output data. The BIDocs methods are the methods that you will most often use to access this data. When you generate a PeopleCode template, these methods are used in that template.

You use BIDocs methods to add input to a Business Interlink object, then you call the Execute method, then you use BIDocs methods to get output from the Business Interlink object.

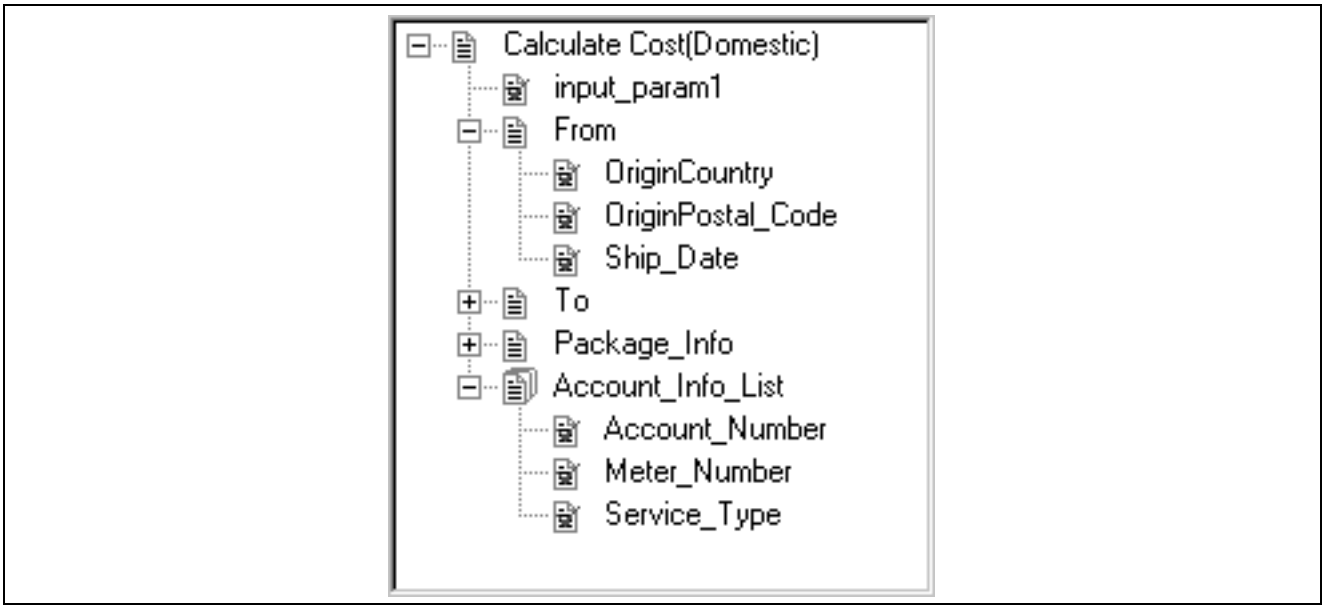
Note. To better illustrate the BIDocs methods, the examples were created from a modified version of a PScustomer plug-in. The input parameter input-param1 and output parameters output_param1 and output_param2_list were added to this example, and the Account_Info input parameter was modified to be a list, Account_Info_List.

See Chapter 6, "Business Interlink Class Methods and Properties," Supporting Standard Input/Output with BIDocs Methods, page 38.

You can think of the Input Doc and the Output Doc that stores the hierarchical data as a tree, with a root doc, node docs, and values.

Using Input Docs

The following shows an example of an Input Doc:



Example Input Doc

The *root doc* is Calculate Cost(Domestic). A root doc can contain both values and node docs.

The *node docs* are From, To, Package_Info and Account_Info_List. Each node can contain both values and child nodes. Node docs can be further described as the following:

- *Simple node docs* have only one set of values for a single instance of a root doc. From, To, Package_Info are all simple node docs.
- *List node docs* can contain more than one set of values for a single instance of a root doc. Account_Info_List is a list node doc.

The *values* in the From node are OriginCountry, OriginPostal_Code and Ship_Date. The value within the root node is input_param1. Notice that there are values both within the root doc and within node docs.

Business Interlinks support Input Docs with the following methods.

- GetInputDocs
- AddDoc
- AddValue
- AddNextDoc

The **GetInputDocs** method creates an Input Doc and returns a reference to its root doc. From the above example, it returns a reference to Calculate Cost(Domestic).

Use the **AddDoc** BIDocs method to return a reference to the node docs. From the above example, you would use AddDoc to access the From, To, Package_Info and Account_Info_List node docs. If any of these nodes contain nodes, you use AddDoc to access those as well.

Use the **AddValue** BIDocs method to set values. From the above example, you would use AddValue to set the value for input_param1, and for the From node, to set the values OriginCountry, OriginPostal_Code, and Ship_date. You must call AddDoc on a node before you can call AddValue for its values.

Use the **AddNextDoc** BIDocs method to access the following:

- Use AddNextDoc to return a reference to the next root doc.

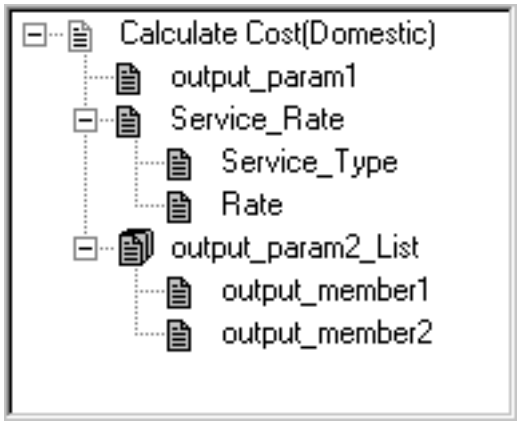
- If a node doc is a list, use AddNextDoc to return a reference to the next node doc in the list.

The following code example sets values for the node doc From, which is a simple node doc. It also sets the values for Account_Info_List, which is a list node doc.

```
&Calc_Input = &QE_COST.GetInputDocs("");

&FromDoc = &Calc_Input.AddDoc("From");
&ret = &FromDoc.AddValue("OriginCountry", "United States");
&ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
&ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

&Account_Doc = &Calc_Input.AddDoc("Account_Info_List");
&ret = &Account_Doc.AddValue("Account_Number", "CT-8001");
&ret = &Account_Doc.AddValue("Meter_Number", &METER);
&ret = & Account_Doc.AddValue("Service_Type", &MODE);

/* add next set of values in list */

&Account_Doc = &Account_Doc.AddNextDoc();
&ret = &Account_Doc.AddValue("Account_Number", "CT-8002");
&ret = &Account_Doc.AddValue("Meter_Number", &METER);
&ret = &Account_Doc.AddValue("Service_Type", &MODE);
```

Using Output Docs

The following shows an example of an Output Doc:



Example Output Structure

The *root doc* is Calculate Cost(Domestic). A root doc can contain both values and node docs.

The *node docs* are Service_Rate and output_param2_List. Each node can contain both values and child nodes. Node docs can be further described as the following:

- *Simple node docs* have only one set of values for a single instance of a root doc. Service_Rate is a simple node doc.
- *List node docs* can contain more than one set of values for a single instance of a root doc. output_param2_List is a list node doc.

The *values* in the Service_Rate node are Service_Type, Rate, and in the output_param2_List node, output_member1 and output_member2, and within the root node, output_param1.

Business Interlinks support hierarchical output structures with the following methods.

- GetOutputDocs
- GetDoc
- GetNextDoc
- GetPreviousDoc
- GetStatus
- GetCount
- MoveToDoc

The **GetOutputDocs** method creates an Output Doc and returns a reference to its root doc. From the above example, it returns a reference to Calculate Cost(Domestic).

Use the **GetDoc** BIDocs method to return a reference to the node docs. From the above example, you would use GetDoc to access the Service_Rate or output_param2_List node docs. If any of these nodes contain nodes, you use GetDoc to access those as well.

Use the **GetValue** BIDocs method to retrieve the values. From the above example, you would use GetValue to retrieve the values for output_param1, and for the Service_Rate node, to get the values Service_Type, Rate, output_member1 and output_member2. You must call GetDoc on a node before you can call GetValue for its values.

Use the **GetNextDoc** and **GetPreviousDoc** BIDocs methods to access the following:

- Use GetNextDoc/GetPreviousDoc to return a reference to the next/previous root doc.
- If a node doc is a list, use GetNextDoc/GetPreviousDoc to return a reference to the next/previous node doc in the list.

Use the **GetCount** method to return either the number of docs in a list node doc, or the number of root docs. In the example, you can count the number of Calculate Cost(Domestic) nodes or the number of output_param2_list nodes.

Use the **MoveToDoc** method to move to either a particular doc at the root level or to a list node doc. In the example, you can move to a Calculate Cost(Domestic) node or to an output_param2_list node.

The following code example gets values for the node docs Service Rate, which is a simple node doc. It also sets the values for output_param2_List, which is a list node doc.

```
&Calc_Output = &QE_COST.GetOutputDocs("");

&Service_Rate_Doc = &Calc_Output.GetDoc("Service_Rate");
&ret = &Service_Rate_Doc.GetValue("Service_Type", &Type);
&ret = &Service_Rate_Doc.GetValue("Rate", &Rate);

&Out_Param_Doc = &Calc_Output.GetDoc("output_param2_List");
&ret = &Out_Param_Doc.GetValue("output_member1", &value1);
&ret = &Out_Param_Doc.GetValue("output_member2", &value2);

/* get next set of values in list */
```

```
&Account_Doc = &Out_Param_Doc.AddNextDoc();
&ret = &Out_Param_Doc.GetValue("output_member1", &value3);
&ret = &Out_Param_Doc.GetValue("output_member2", &value4);
```

Supporting Batch Input and Output

The methods discussed in this chapter, except for BulkExecute, add input values to the Business Interlink object one set, or row, at a time. The call to the Business Interlink plug-in occurs only once. All the input data is passed with the single **Execute**. All output is returned as batch as well. The methods then get the output values one set, or row, at a time.

If you're sending a large amount of data to the input buffers, instead of adding one input row at a time, you might write the data to a staging table, then use the **BulkExecute** method. This method automatically executes; that is, you don't have to use the Execute method. It also automatically fills the output record specified with the method with all the output values in every row in the output buffer if you've specified an output record.

Supporting Rowsets

If your data is mapped into rowsets, you may want to use the **InputRowset** method. This method will take a standard rowset object to populate the inputs for the Business Interlink object. You can use the **FetchIntoRowset** method to repopulate the rowset with output from the Business Interlink object.

Using the Flat Table Input/Output Methods

If your data is in a flat table structure, you can use the flat table methods. **AddInputRow** adds rows of input to the Business Interlink object; **FetchNextRow** fetches rows of output from the Business Interlink object.

Using the State of a Business Interlink Object

Your PeopleSoft Business Interlink API should be stateless, that is, if you want to save information from one call of the Business Interlink object to the next, you will have to do it yourself by writing the relevant information to the database. If you use the **Execute** method more than once within a single PeopleCode event (that is, if you have the **Execute** method in some sort of loop) the state will be preserved. Once you leave the event, any state associated with the Business Interlink object is lost.

You should only create one Business Interlink object, that is, you should only use the **GetInterlink** function once. After that, you can load it with data, pass the data to the Business Interlink plug-in (via **Execute**) and fetch output data as many times as you need.

Declaring a Business Interlink Object

You should declare a Business Interlink object using the data type Interlink. In a regular PeopleCode program, you can only declare a Business Interlink object as local. However, in an Application Engine program, you can declare a Business Interlink object as global. Instantiating a Business Interlink object as global saves on the significant overhead of reinstantiating a local object for every iteration of PeopleCode called in a loop.

Global Business Interlink objects can only be used in Application Engine PeopleCode programs because PeopleCode that runs on an application server must be stateless.

When a restartable Application Engine program abends, global Business Interlink objects that were instantiated before the last checkpoint are automatically reinstantiated at restart. So the object will be available, even though no call has been made to **GetInterlink** in the restarted process. However, the associated Business Interlink data buffers are *not* recovered, so the Application Engine program must be written such that these buffers are empty whenever a checkpoint is taken.

Business Interlink objects should *not* be declared as global unless they are used in several PeopleCode actions, or in a PeopleCode action that is called in a loop. Only in these instances is the overhead of checkpointing them worthwhile.

Scoping a Business Interlink Object

A Business Interlink object can be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in message subscription PeopleCode, Application Engine PeopleCode, record field PeopleCode, etc.

Using Business Interlink Object Methods

This section describes the PeopleCode methods you use to instantiate a Business Interlink object and execute that object. You can instantiate an instance of a named Business Interlink object using the PeopleCode GetInterlink function and then use methods **(AddInputRow, Execute, FetchNextRow)** on that instance to add data to the object, execute the object, and fetch output data from the object. If any of these methods fail due to error, either an error is displayed and execution of the PeopleCode stops, or the error is logged and processing continues, depending on the setting of the StopAtError property.

The BIDocs methods support hierarchical data. Use them if your input and output is in hierarchical form.

See Chapter 6, "Business Interlink Class Methods and Properties," Using BIDocs Hierarchical Methods, page 65.

AddInputRow

Syntax

AddInputRow(*inputname, value*)

where *inputname* and *value* are in matched pairs, in the form:
inputname1, value1 [*, inputname2, value2*] . . .

Description

The **AddInputRow** method adds a row of input data *(value)* from PeopleCode variables or record fields to the specified input names *(inputname)* for the Business Interlink object executing the method. These must be entered in matched pairs, that is, every input name must be followed by its matching value.

Note. The input **name**, not the input path, of the interface definition is used for this method.

There must be an *inputname* for every input parameter defined in the interface definition used to instantiate the Business Interlink object.

If you specify a record field that is not part of the record the PeopleCode program is associated with, you must use *recname.fieldname* for that *value.*

You can specify default values for every input name in the interface definition (created in the Application Designer.) These values will be used if you generate a PeopleCode template by dragging the interface definition from the Project window in the Application Designer to an open PeopleCode editor window.

See Chapter 4, "Generating the PeopleCode Template," page 23.

Parameters

Parameter	Description
inputname	Specify the input name. There must be one inputname for every input name defined in the interface definition used to instantiate the Business Interlink object.
value	Specify the value for the input name. This can be a constant, a variable, or a record field. Each *value* must be paired with an *inputname*.

Returns

A Boolean value: True if the input values were successfully added. Otherwise, it returns False.

Example

In the following example, the Business Interlink object name is QE_COST, and the input names, such as OriginPostal_Code, are being bound to variables like &ORIGIN, or to literals, like 10 for quantity. The input names could also be bound to record fields.

```
Local Interlink &QE_COST;
&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&QE_COST.AddInputRow("OriginCountry", "United States",
   "OriginPostal_Code", &ORIGIN,
   "Ship_Date", &SHIPDATE,
   "DestCountry", "United States",
   "DestPostal_Code", &DESTINATION,
   "Weight", &WEIGHT,
   "Weight_Type", "LBS",
   "Declared_Value", &VALUE,
   "Account_Number", &ACCOUNT,
   "Meter_Number", &METER,
   "Service_Type", &SVCTYPE);
```

BulkExecute

Syntax

BulkExecute(**RECORD**.*inputrecname* [, **RECORD**.*outputrecname*] [, {*user_process_inst* | *user_operid*}])

Description

The **BulkExecute** method uses the data in the specified record to populate the input buffer, copying *like-named* fields. Then the method executes, and, optionally, fills the record specified by *outputrecname* with data from the Business Interlink output buffer. **BulkExecute** will result in significantly faster performance for transactions that process large amounts of data. Instead of adding one input row at a time, then fetching the values one at a time, you might write the data to a staging table, use the **BulkExecute** method and then read the data from the output table. This would be especially effective in Application Engine programs that process sets of data rather than individual rows.

This method assumes that the names of the fields in the record match the names of the inputs (or outputs) defined in the Business Interlink definition. If there is no field in the *inputrecname* for a Business Interlink input parameter, the parameter's default value is used. If no default is specified, an empty string is passed. It's up to the programmer to ensure that this default value is legitimate.

Fields in the output buffer are matched against the fields in the specified in the output record. You do not have to specify an output record. If you don't specify an output record, the output buffers will not be populated.

Note. Before you use this method, you should flush the record used for output and remove any residual data that might exist in it.

If you specify an output record, and you want to use the output record for error checking, you must add the following fields to your output record:

- RETURN_STATUS
- RETURN_STATUS_MSG

Note. The field names in the output record must match these names exactly.

Then you must mark one or more input parameters as *key fields* in the Business Interlink definition (using the BulkExecute ID check box). The value of these key fields will be copied to the output record, and you can use these fields to match error messages with input (or output) rows.

See Chapter 3, "Designing a Business Interlink Definition," Setting Business Interlink Inputs, page 14.

If you want to order your input (and output) rows, you must add a column called BI_SEQ_NUM to both the input and output table.

Note. The field name in your input and output records must match this name exactly.

The input rows will be read in the order of numbers in BI_SEQ_NUM, and the output rows will be generated using the same order number.

This method automatically executes, so you don't need to use the **Execute** method with this method.

Whether this method halts on execution or not depends on the setting of the StopAtError configuration parameter. The default value is True, that is, stop if the error number returned is something other than a 1 or 2. This configuration parameter must be set before using the **BulkExecute** method.

See Chapter 6, "Business Interlink Class Methods and Properties," BulkExecute, page 44 and Chapter 3, "Designing a Business Interlink Definition," Setting Business Interlink Inputs, page 14.

Parameters

Parameter	Description
RECORD.inputrecname	Specify a record (SQL table) that contains the data you want to use to populate the input buffer of the Business Interlink. If this record has an integer column named BI_SEQ_NUM, that column must contain a sequence number which corresponds to the order in which each row of the SQL table will be read by **BulkExecute**.
RECORD.outputrecname	Specify a record (SQL table) that will hold the data that populates the output buffer of the Business Interlink. This is optional; when used, it is often used to error check the input. If *outputrecname* is NONE, **BulkExecute** will not fill an output record. If this record has an integer column named BI_SEQ_NUM, that column contains a sequence number which corresponds to the order in which each row of the SQL table was written by **BulkExecute**; there will be a one-to-one correspondence with the numbers in the BI_SEQ_NUM column in *RECORD.inputrecname*.
user_process_inst | user_operid	This is an optional parameter that allows either different Application Engine programs or different clients to populate the same *RECORD.outputrecname* at the same time. For *user_process_inst*, the parameter takes an integer. *RECORD.outputrecname* must have a PROCESS_INSTANCE field. The PROCESS_INSTANCE field is used to identify the Application Engine program that is using this Business Interlink. You can use the %PROCESS_INSTANCE variable to populate *user_process_inst*. For *user_operid*, the parameter takes a string. *RECORD.outputrecname* must have an OPERID field. The OPERID field is used to identify the client who is using this Business Interlink. You can use the %OPERID variable to populate *user_operid*.

Returns

A number representing the return status.

Number	Meaning
1	The Business Interlink object executes successfully.
2	The Business Interlink object failed to execute.
3	Transaction failed
4	Query failed
5	Missing criteria
6	Input mismatch
7	Output mismatch
8	No response from server

Number	Meaning
9	Missing parameter
10	Invalid username
11	Invalid password
12	Invalid server name
13	Connection error
14	Connection refused
15	Timeout reached
16	Unequal lists
17	No data for output
18	Output parameters empty
19	Driver not found
20	Internet connect error
21	XML parser error
22	XML deserialize
201	Warning: Field too large (execute succeed)
202	Ignore sign number (execute succeed)
203	Convert Float to integer (execute succeed)

Example

Here are two PeopleSoft records that could be used as input and output records for BulkExecute. The key fields in the input record, which need to have the BulkExecute ID check box checked in the Application Designer, are QE_RP_PO_NUMBER and QE_RP_SITENAME. These key fields are used both for input and output.

The following record is the input record.

QE_RP_PO (Record)

Record Fields | Record Type

Num	Field Name	Type	Len	Format	H	Short Name	Long Name
1	QE_RP_PO_NUMBER	Char	40	Mixed		QE_RP_PONUMBER	QE_RP_PONUMBER
2	QE_RP_PODATE	DtTm	26			QE_RP_PODATE	QE_RP_PODATE
3	QE_RP_SITENAME	Char	15	Mixed		SITE_NAME	SITE_NAME
4	QE_RP_VENDORNAME	Char	30	Mixed		QE_RP_VENDORNAM	QE_RP_VENDORNAME

Example input record for BulkExecute

The following record is the output record.

QE_RP_PO_OUT1 (Record)

Record Fields | Record Type

Num	Field Name	Type	Len	Format	H	Short Name	Long Name
1	QE_RP_PO_NUMBER	Char	40	Mixed		QE_RP_PONUMBER	QE_RP_PONUMBER
2	QE_RP_SITENAME	Char	15	Mixed		SITE_NAME	SITE_NAME
3	RETURN_STATUS	Sign	3			RETURN_STATUS	RETURN_STATUS
4	RETURN_STATUS_MSG	Char	254	Mixed		RETURN_STAT_MSG	RETURN_STATUS_MSG

Example output record for BulkExecute

Here are the corresponding inputs and outputs for a Business Interlink that has had its inputs and outputs renamed to match the field names in the records. The inputs and outputs have been renamed where necessary to match the record field names.

Following are the inputs.

QE_RP_CREATEPO1 (Business Interlink)

Input | Output | Test | Settings

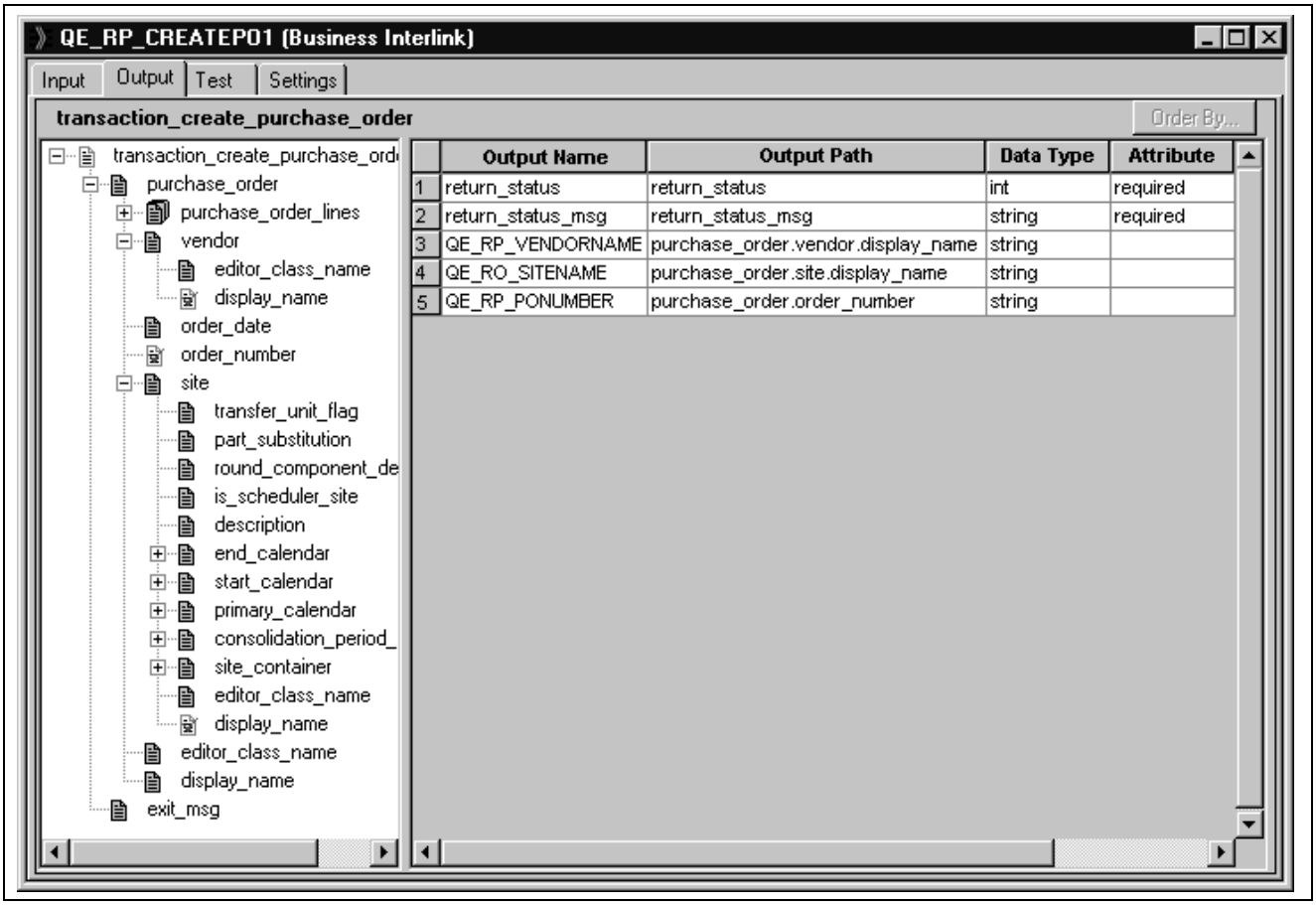
transaction_create_purchase_order

- transaction_create_purcha
  - class_name
  - order_date
  - vendor_name
  - po_number
  - site_name

	Input Name	Input Path	Default	Data Typ	Attribute	BulkExec Id	Path Ir
1	class_name	class_name	Purchase_Or	string	required	☐	string
2	QE_RP_PODATE	order_date		string	required	☐	string
3	QE_RP_VENDORNAME	vendor_name		string	required	☐	string
4	QE_RP_PO_NUMBER	po_number		string	required	☒	string
5	QE_RP_SITENAME	site_name		string	required	☒	string

Example inputs for BulkExecute

Following are the outputs.



Example outputs for BulkExecute

The PeopleCode program using these records could contain the following code to run **BulkExecute**.

```
Local Interlink &CREATE_PO;
Local number &EXECRSLT;

&CREATE_PO = GetInterlink(INTERLINK.QE_RP_CREATEPO1);

/* The next three lines set configuration parameters, which are not shown in the?
 previous examples. */
&CREATE_PO.SERVER_NAME = "bc1";
&CREATE_PO.RSERVER_HOST = "4.3.4.3";
&CREATE_PO.RSERVER_PORT = "2200";


&EXECRSLT = &CREATE_PO.BulkExecute(RECORD.QE_RP_PO, RECORD.QE_RP_PO_OUT1);
```

Clear

Syntax

Clear()

Description

The **Clear** method clears the input and output buffers. If you're using Business Interlinks in a batch program, every time after you use the **Execute** method, you want to use the **Clear** method to flush out the input and output buffers.

Parameters

None.

Returns

None.

Example

```
For &n = 1 to x
   &myinterlink.AddInputRow("input1", value1, "input2", value2);
   &myinterlink.Execute();
   &myinterlink.FetchNextRow("output1", value1, "output2", value2);
      /* do processing on data */
   &myinterlink.Clear();
   &n = &n + 1;
End-for;
```

Execute

Syntax

Execute()

Description

When a Business Interlink object executes the **Execute** method, the transaction associated with the Business Interlink object is executed.

If there is only one row, after appropriate substitution is made, the transaction is executed only once. Otherwise, the data is "batched up" and sent once. You only have to call **Execute** once to execute all the rows of the input buffer.

Generally you would only use the **Execute** method after using the **AddInputRow** or **InputRowset** methods. If you generate a PeopleCode template by dragging the Business Interlink definition from the Project window in the Application Designer to an open PeopleCode editor window, the usual order of the execution of methods is shown in the template.

See Chapter 4, "Generating the PeopleCode Template," page 23.

If you're using Business Interlinks in a batch program, every time after you use the **Execute** method, you want to use the **Clear** method to flush out the input and output buffers.

Whether this method halts on execution or not depends on the setting of the StopAtError configuration parameter. The default value is True, that is, stop if the error number returned is something other than a 1 or 2. This configuration parameter must be set before using the **Execute** method.

Parameters

None.

Returns

A number representing the return status.

Number	Meaning
1	The Business Interlink object executes successfully.
2	The Business Interlink object failed to execute.
3	Transaction failed
4	Query failed
5	Missing criteria
6	Input mismatch
7	Output mismatch
8	No response from server
9	Missing parameter
10	Invalid username
11	Invalid password
12	Invalid server name
13	Connection error
14	Connection refused
15	Timeout reached
16	Unequal lists
17	No data for output
18	Output parameters empty

Number	Meaning
19	Driver not found
20	Internet connect error
21	XML parser error
22	XML deserialize

Example

```
Local number &EXECRSLT;
. . .
&EXECRSLT = &SRA_ALL_1.Execute();
If (&EXECRSLT <> 1) Then
   /* The instance failed to execute */
   /* Do Error Processing */
End-If;
```

Using New Status Codes for the pshttpenable (Generic) Runtime Plug-In

There are new status codes for the execute method relating to using an XML design-time plug-in that uses the generic Business Interlink (pshttpenable runtime plug-in). These status codes cover when a business interlink is called to pull in external data from a non-valid address.

To write your PeopleCode to take the status codes into account:

1. When you use the PeopleCode BIDocs Execute method with a Business Interlink definition created from such a design-time plug-in, check the return status for success (=1). Failure at this point is rare, and is most likely to be failure to find the Business Interlink object.
2. After the Execute call, use the GetValue method on the output document to get the values of the outputs return_status and return_status_msg.
3. If return_status = 200, the HTTP completed successfully. However, you should check the response to see if it has proper content; a successful HTTP does not mean that the transaction successfully returned content. Within the output doc, run GetDoc and GetStatus on a doc that you know must be returned. Many merchants provide an error code doc; use this if available, and read its value with GetValue to see if the service was able to complete the transaction.
4. If return_status = -1, the URL you provided is likely to be malformed. If return_status is 1, you used a simulated response file instead of doing HTTP. These status codes mean that no HTTP was performed. If return_status is 400 or greater, the HTTP failed in some way; you can look up these HTTP status codes on the following website: http://www.w3.org/Protocols/rfc2616/rfc2616.html. There is also a table at the end of this section containing some of these status codes.

Following is a pseudocode example of performing these steps.

```
&MyBI = GetInterlink(Interlink.SomeBI);
/* change the config parameters if needed*/
&MyBI.Method = "POST";
/*get the input doc */
```

```
&inDoc = &MyBI.GetInputDocs("");
/*create docs and input values*/
&ReqDoc = &inDoc.AddDoc("postreq");
&ret = &postReqDoc.AddValue("userid", "24");
/* ===> Step 1: Check the status of Execute. */
&EXECRSLT = &QE_GETSV_1.Execute();
If (&EXECRSLT <> 1) Then
   /* The instance failed to execute */
Else
/* ===> Step 2: Get the value of return_status. */
   &ret = &outDoc.GetValue("return_status", &return_status);
   if(&ret = 200) then
      &outDoc = &MyBI.GetOutputDocs("");
/* ===> Step 3: Get a document (error) within the output doc (response), and if?
 it is an error code supplied by the merchant, get the?
 value of the error code and preform appropriate action. */
      &errorDoc = &outDoc.GetDoc("response.error");
      &docret = &errorDoc.GetStatus();
      if (&docret =0) then
         &errorDoc.GetValue("errorcode", &errorcode);
         if (no error) then
             ............
            /*Get Docs and values*/
            &userDoc = &outDoc.GetDoc("postreqresponse.candidates.user");
            &userDoc.GetValue("conid", &conid);
         else
            /* flag the error and take appropriate action */
         End-if;
      else
         /* error processing for not response doc returned */
      end-if;
   else
      /* ===> Step 4: process for HTTP status codes. */

   End-if;
end-if;
```

The following table contains the values for return_status and return_status_msg.

return_status	return_status_msg
0	Response read from file
-1	Internet access failed
-1	Setting Output parameters failed
-1	Invalid Inputs

return_status	return_status_msg
-1	URL Error
-1	Error sending request
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	Reset Content
205	Partial Content
400	Bad Request
404	Not Found
405	Method Not Allowed
500	Internal Server Error

The values of 0 and 200-206 for return_status mean a response was successfully read. Values of −1 and 0 are specific to pshttpenable; values of 200 or greater are standard Hypertext Transfer Protocol. You can see the following URL for return_status values of 200 or greater.

FetchIntoRecord

Syntax

```
FetchIntoRecord(RECORD.recname [, {user_process_inst | user_operid]})
```

Description

The **FetchIntoRecord** method copies the data from the output buffer into the specified record (SQL table), copying like-named fields. This method assumes that the names of the fields in the record match the names of the outputs defined in the Business Interlink definition.

You can use the **FetchIntoRecord** method to perform a transaction on a large amount of data that you want to receive from your system. Instead of executing your Business Interlink and then fetching one output row at a time, you might execute your Business Interlink, then use the **FetchIntoRecord** method to write the data returned from the Business Interlink to a staging table.

Note. Before you use this method, you should flush the record used for output and remove any residual data that might exist in it.

If your data is hierarchical, i.e., in a rowset, and you want to preserve the hierarchy, use **FetchIntoRowset** instead of this method.

If you want to order your output rows, you must add the following column to the record: BI_SEQ_NUM.

This column will be populated with a sequence number that corresponds to the order in which each row of the record was written to by the method.

Parameters

Parameter	Description
RECORD.recname	Specify a record (SQL table) that you want to populate with data from the output buffers. If this record has an integer column named BI_SEQ_NUM, that column contains a sequence number which corresponds to the order in which each row of the SQL table was written by **FetchIntoRecord**.
user_process_inst | user_operid	This is an optional parameter that allows either different Application Engine programs or different clients to populate the same *RECORD.outputrecname* at the same time.
	For *user_process_inst*, the parameter takes an integer. *RECORD.outputrecname* must have a PROCESS_INSTANCE field. The PROCESS_INSTANCE field is used to identify the Application Engine program that is using this Business Interlink. You can use the %PROCESS_INSTANCE variable to populate *user_process_inst*.
	For *user_operid*, the parameter takes a string. *RECORD.outputrecname* must have an OPERID field. The OPERID field is used to identify the client who is using this Business Interlink. You can use the %OPERID variable to populate *user_operid*.

Returns

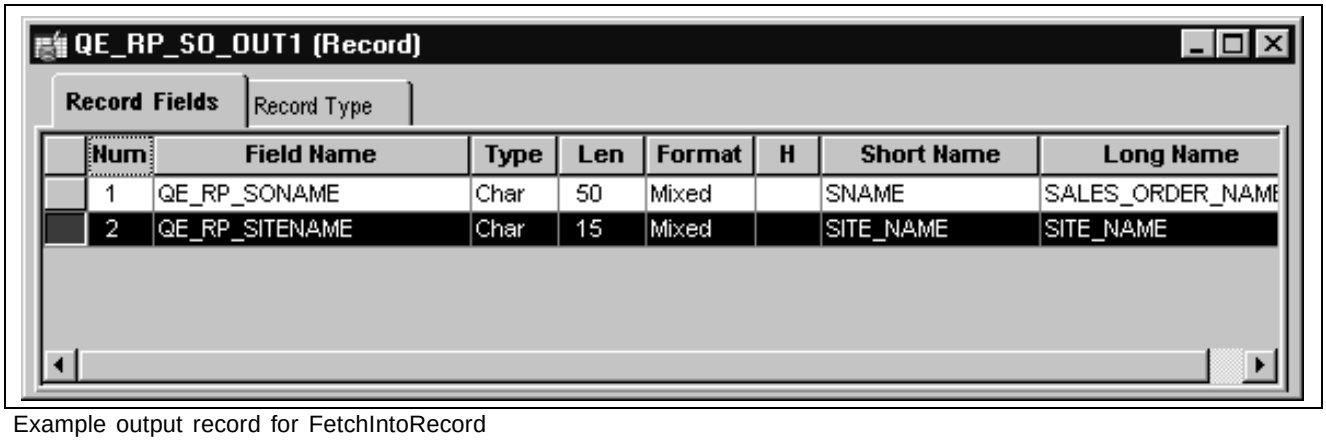
Number of rows fetched if one or more non-empty rows are returned. If an empty row is returned, that is, if every field is empty except the return_status and return_status_msg fields, this method returns one of the following error messages:

Number	Meaning
1	NoInterfaceObject
2	ParamCountError
3	IncorrectParameterType
4	NoColumnForOutputParm
5	NoColumnForInputParm

Number	Meaning
6	BulkInsertStartFailed
7	BulkInsertStopFailed
8	CouldNotCreateSelectCursor
9	CouldNotCreateInsertCursor
10	CouldNotDestroySelectCursor
11	CouldNotDestroyInsertCursor
12	InputRecordDoesNotExist
13	OutputRecordDoesNotExist
14	ContainEmptyRow
15	SQLExecError
201	FieldTooLarge
202	IgnoreSignNumber
203	ConvertFloatToInt

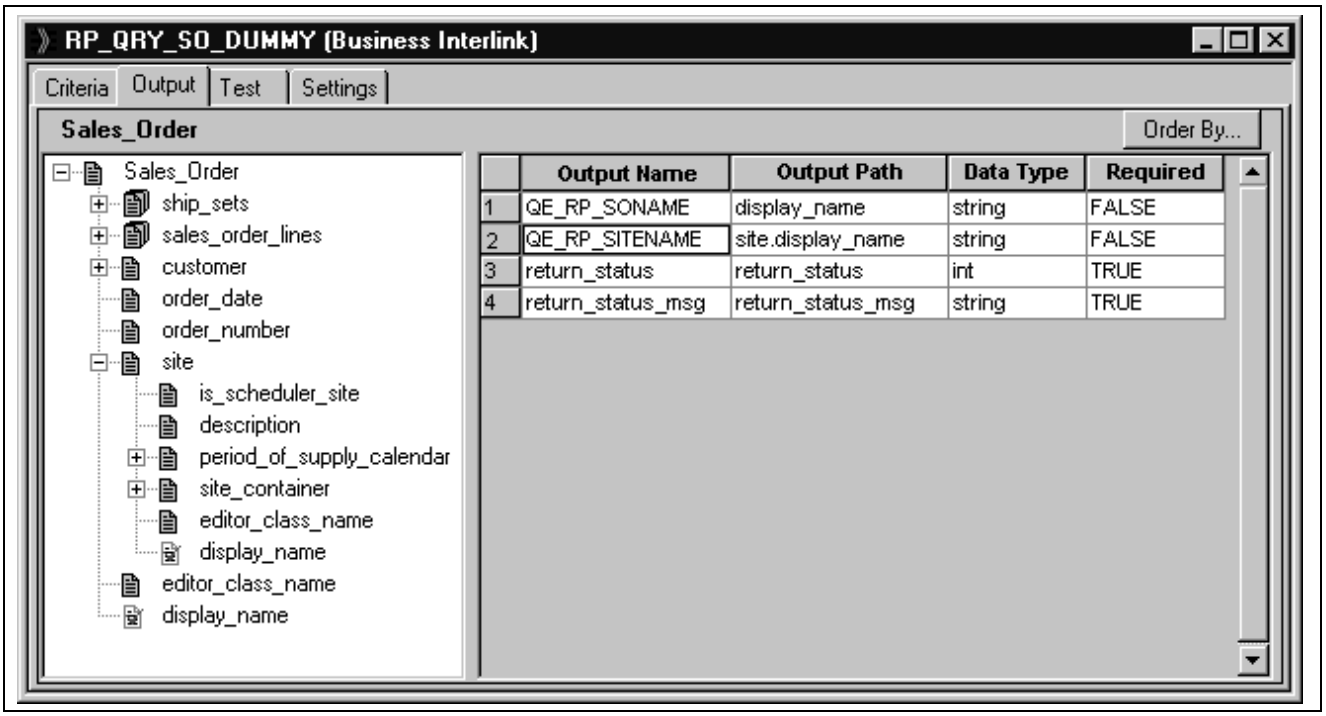
Example

Here is a PeopleSoft record that could be used as an output record for FetchIntoRecord.



Example output record for FetchIntoRecord

Here are the corresponding outputs for a Business Interlink that has had its outputs renamed to match the field names in the records.



Example outputs for FetchIntoRecord

The PeopleCode program using these records could contain the following code to run **FetchIntoRecord**.

```
Local Interlink &RP_QRY_SO_EXAMPLE_1;
Local number &RSLT;

&RP_QRY_SO_EXAMPLE_1 = GetInterlink(INTERLINK.RP_QRY_SO_EXAMPLE);

/* ===> Add Criteria inputs (RHS): */
&RP_QRY_SO_EXAMPLE_1.AddInputRow("IN1_customer.display_name",  "Joe Blow" ,?
 "IN2_order_number",  "199" , "IN3_order_date",  "09/18/99" ,?
 "IN4_order_date",  "10/18/96"  );

&EXECRSLT = &RP_QRY_SO_EXAMPLE_1.Execute();
If ( &EXECRSLT <> 1 ) Then
   /* The instance failed to execute */
Else
   &RSLT =
      &RP_QRY_SO_EXAMPLE_1.FetchIntoRecord(RECORD.QE_RP_SO_OUT1);
End-If; /* If NOT &RSLT ... */
```

FetchIntoRowset

Syntax

FetchIntoRowset(*&Rowset*)

Description

The **FetchIntoRowset** method copies the data from the output buffer into the specified rowset, copying *like-named* fields. This method assumes that the names of the fields in the rowset match the names of the outputs defined in the Business Interlink definition, *and* that the structure is the same. You can use the **FetchIntoRowset** method to repopulate the rowset with output from the Business Interlink object.

Note. *Before* you use this method, you should flush the rowset used for output and remove any residual data that might exist in it.

Use this method only if you have a hierarchical data structure and you want to preserve the hierarchy. Otherwise, use **BulkExecute** or **FetchIntoRecord**.

Parameters

Parameter	Description
&Rowset	Specify rowset object that you want to populate with data from the output buffers. This must be an existing, instantiated rowset.

Returns

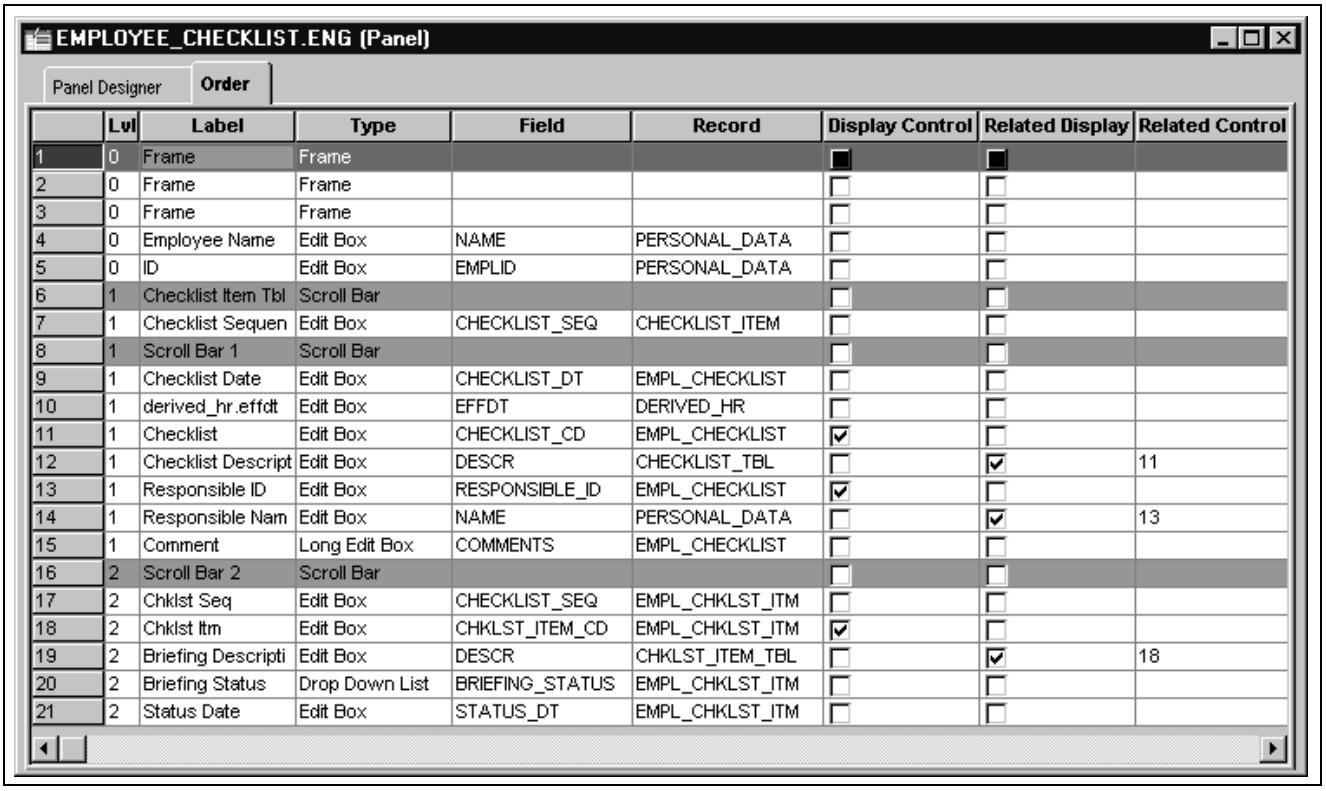
Number of rows fetched if one or more non-empty rows are returned. If an empty row is returned, that is, every if field is empty except the return_status and return_status_msg fields, this method returns one of the following error messages:

Number	Meaning
1	NoInterfaceObject
2	ParamCountError
3	IncorrectParameterType
4	NoColumnForOutputParm
5	NoColumnForInputParm
6	BulkInsertStartFailed
7	BulkInsertStopFailed
8	CouldNotCreateSelectCursor
9	CouldNotCreateInsertCursor
10	CouldNotDestroySelectCursor

Number	Meaning
11	CouldNotDestroyInsertCursor
12	InputRecordDoesNotExist
13	OutputRecordDoesNotExist
14	ContainEmptyRow
15	SQLExecError
201	FieldTooLarge
202	IgnoreSignNumber
203	ConvertFloatToInt

Example

The example uses the rowset on level 1 from the EMPLOYEE_CHECKLIST page. A PeopleCode event running on level 0 in that page can access the child rowset (level 1), shown below from Scroll Bar 1 to Scroll Bar 2.



	Lvl	Label	Type	Field	Record	Display Control	Related Display	Related Control
1	0	Frame	Frame			■	■	
2	0	Frame	Frame			☐	☐	
3	0	Frame	Frame			☐	☐	
4	0	Employee Name	Edit Box	NAME	PERSONAL_DATA	☐	☐	
5	0	ID	Edit Box	EMPLID	PERSONAL_DATA	☐	☐	
6	1	Checklist Item Tbl	Scroll Bar			☐	☐	
7	1	Checklist Sequen	Edit Box	CHECKLIST_SEQ	CHECKLIST_ITEM	☐	☐	
8	1	Scroll Bar 1	Scroll Bar			☐	☐	
9	1	Checklist Date	Edit Box	CHECKLIST_DT	EMPL_CHECKLIST	☐	☐	
10	1	derived_hr.effdt	Edit Box	EFFDT	DERIVED_HR	☐	☐	
11	1	Checklist	Edit Box	CHECKLIST_CD	EMPL_CHECKLIST	☑	☐	
12	1	Checklist Descript	Edit Box	DESCR	CHECKLIST_TBL	☐	☑	11
13	1	Responsible ID	Edit Box	RESPONSIBLE_ID	EMPL_CHECKLIST	☑	☐	
14	1	Responsible Nam	Edit Box	NAME	PERSONAL_DATA	☐	☑	13
15	1	Comment	Long Edit Box	COMMENTS	EMPL_CHECKLIST	☐	☐	
16	2	Scroll Bar 2	Scroll Bar			☐	☐	
17	2	Chklst Seq	Edit Box	CHECKLIST_SEQ	EMPL_CHKLST_ITM	☐	☐	
18	2	Chklst Itm	Edit Box	CHKLST_ITEM_CD	EMPL_CHKLST_ITM	☑	☐	
19	2	Briefing Descripti	Edit Box	DESCR	CHKLST_ITEM_TBL	☐	☑	18
20	2	Briefing Status	Drop Down List	BRIEFING_STATUS	EMPL_CHKLST_ITM	☐	☐	
21	2	Status Date	Edit Box	STATUS_DT	EMPL_CHKLST_ITM	☐	☐	

EMPLOYEE_CHECKLIST

EMPL_CHECKLIST is the primary database record for that scrollbar on the EMPLOYEE_CHECKLIST page. The following PeopleCode access the level 1 rowset using EMPL_CHECKLIST. The Business Interlink object name is QE_BI_EMPL_CHECKLIST. This Business Interlink object uses the level 1 rowset as its input and its output.

The **InputRowset** method uses this rowset as input for QE_BI_EMPL_CHECKLIST. Then a blank duplicate of the rowset is created with CreateRowset, and then the output of QE_BI_EMPL_CHECKLIST is fetched into the blank rowset with **FetchIntoRowset**.

```
&MYROWSET = GetRowset(SCROLL.EMPL_CHECKLIST);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.InputRowset(&MYROWSET);
&RSLT = &QE_BI_EMPL_CHECKLIST.Execute();
/* do some error processing */
&WorkRowset = CreateRowset(&MYROWSET);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.FetchIntoRowset(&WorkRowset);

If &ROWCOUNT = 0 Then
   /* do some error processing */
Else
   /* Process the rowset from QE_BI_EMPL_CHECKLIST. Check it for errors. */
   For &I = 1 to &WorkRowset.RowCount
      For &K = 1 to &WorkRowset(&I).RecordCount
         &REC = &WorkRowset(&I).GetRecord(&K);
         &REC.ExecuteEdits();
         For &M = 1 to &REC.FieldCount
            If &REC.GetField(&M).EditError Then
               /* there are errors */
               /* do other processing */
            End-If;
         End-For;
      End-For;
   End-for;
End-if;
```

FetchNextRow

Syntax

FetchNextRow(*outputname, value*)

where *outputname* and *value* are in matched pairs, in the form:

outputname1, value1 [*, outputname2, value2*] . . .

Description

After the Business Interlink object executes the method **Execute**, the **FetchNextRow** method can be used to retrieve a row of output and store the values of the output name *(outputname)* to PeopleCode variables or record fields *(value).* These must be entered in matched pairs, that is, every output name must be followed by its matching value.

Note. The output *name*, not the output path of the interface definition, is used for this method.

There must be an *outputname* for every output name defined in the interface definition used to instantiate the Business Interlink object.

If you specify a record field that is not part of the record that the PeopleCode program is associated with, you must use *recname.fieldname* for that *value*.

Parameters

Parameter	Description
outputname	Specify the output name. There must be one *outputname* for every output name defined in the interface definition used to instantiate the Business Interlink object.
value	Specify the value for the output name. This can be a variable or a record field. Each *value* must be paired with an *outputname*.

Returns

A Boolean value: True if the row of output parameters was fetched. Otherwise, it returns False.

Example

In the following example, the Business Interlink object name is QE_COST, and the output names, such as *Service_Type*, are being bound to variables, such as &SVCTYPESTR.

```
&RSLT = &QE_COST.FetchNextRow(
   "Service_Type", &SVCTYPESTR,
   "Rate", &RATESTR,
   "return_status", &STATUSSTR,
   "return_status_msg", &MSGSTR);
```

GetInputDocs

Syntax

```
GetInputDocs("")
```

Description

The **GetInputDocs** method gets the top BIDocs input document at the root level for a Business Interlink object. This is a hierarchical structure that will contain the values for the inputs for this Business Interlink object. The methods that you use to put the input values into this document are the BIDocs methods.

See Chapter 6, “Business Interlink Class Methods and Properties,” Using BIDocs Hierarchical Methods, page 65.

Parameters

None.

Returns

A BIDocs input document. This is the document at the top of the root level of the BIDocs input document for a Business Interlink object.

Example

```
Local Interlink &QE_COST;
```

```
Local BIDocs &CalcCostOut, &CalcCostIn;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
&CalcCostIn = &QE_COST.GetInputDocs("");

/* You can now insert the input values and execute the Business Interlink object. */
```

GetOutputDocs

Syntax

GetOutputDocs("")

Description

The **GetOutputDocs** method gets the top BIDocs output document at the root level for a Business Interlink object. This is a hierarchical structure that contains the values for the outputs for this Business Interlink object. The methods that you use to get output values from this document are the BIDocs methods.

See Chapter 6, "Business Interlink Class Methods and Properties," Using BIDocs Hierarchical Methods, page 65.

Parameters

None.

Returns

A BIDocs output document. This is the document at the top of the root level of the BIDocs output document for a Business Interlink object.

Example

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn, &CalcCostOut;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&CalcCostOut = &QE_COST.GetOutputDocs("");

/* You can now execute the Business Interlink object and get the output values. */
```

InputRowset

Syntax

InputRowset(*&Rowset*)

Description

The **InputRowset** method uses the data in the specified rowset to populate the input buffer, copying *like-named* fields in the appropriate structure. This method assumes that the names of the fields in the rowset match the names of the inputs defined in the Business Interlink definition, *and* that the structure is the same.

Use this method only if you have a hierarchical data structure and you want to preserve the hierarchy. Otherwise, use **BulkExecute** or **FetchIntoRecord**.

Parameters

Parameter	Description
&Rowset	Specify an existing, instantiated rowset object.

Returns

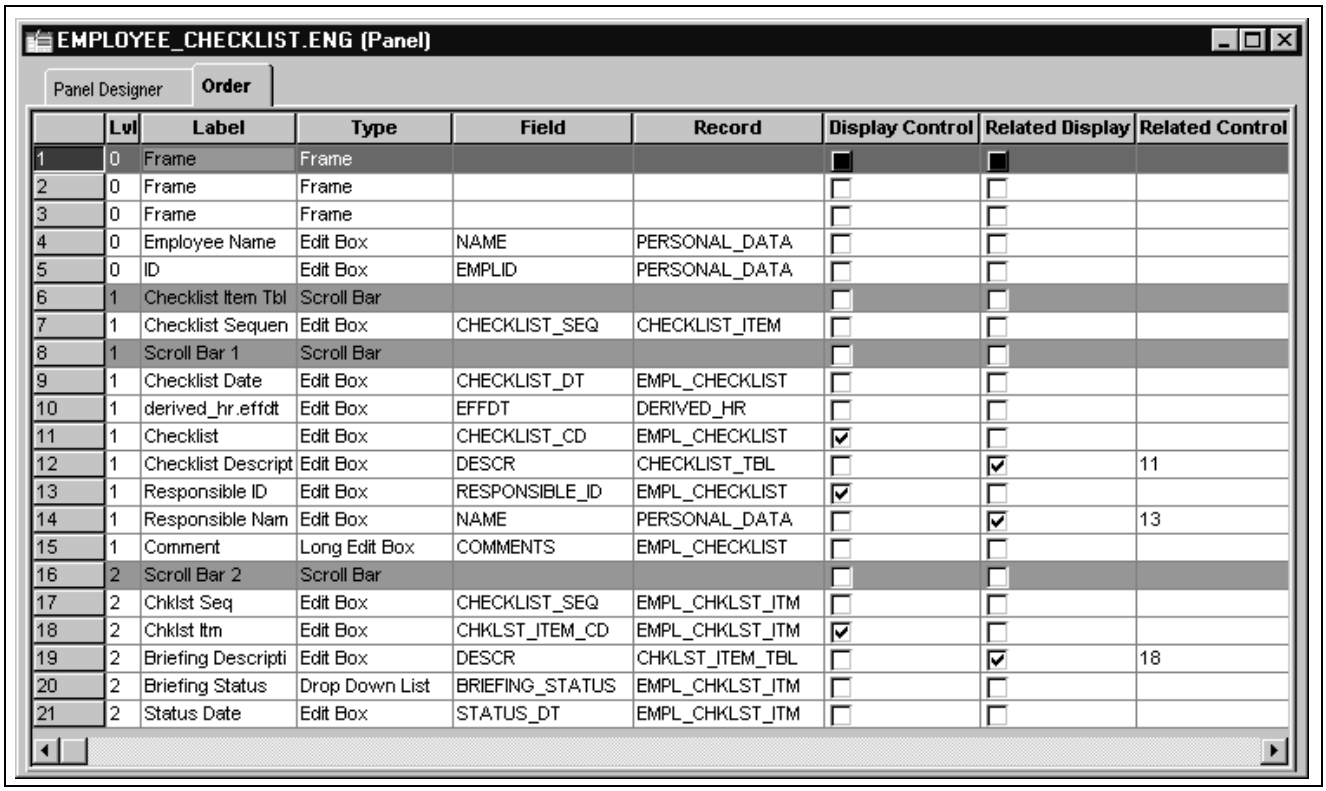
An optional value: the number of rows inserted into the output buffer.

If an empty row is returned, that is, every if field is empty except the return_status and return_status_msg fields, this method returns one of the following error messages:

Number	Meaning
1	NoInterfaceObject
2	ParamCountError
3	IncorrectParameterType
4	NoColumnForOutputParm
5	NoColumnForInputParm
6	BulkInsertStartFailed
7	BulkInsertStopFailed
8	CouldNotCreateSelectCursor
9	CouldNotCreateInsertCursor
10	CouldNotDestroySelectCursor
11	CouldNotDestroyInsertCursor
12	InputRecordDoesNotExist
13	OutputRecordDoesNotExist
14	ContainEmptyRow
15	SQLExecError

Example

The example uses the rowset on level 1 from the EMPLOYEE_CHECKLIST page. A PeopleCode event running on level 0 in that page can access the child rowset (level 1), shown below from Scroll Bar 1 to Scroll Bar 2.



	Lvl	Label	Type	Field	Record	Display Control	Related Display	Related Control
1	0	Frame	Frame			■	■	
2	0	Frame	Frame			☐	☐	
3	0	Frame	Frame			☐	☐	
4	0	Employee Name	Edit Box	NAME	PERSONAL_DATA	☐	☐	
5	0	ID	Edit Box	EMPLID	PERSONAL_DATA	☐	☐	
6	1	Checklist Item Tbl	Scroll Bar			☐	☐	
7	1	Checklist Sequen	Edit Box	CHECKLIST_SEQ	CHECKLIST_ITEM	☐	☐	
8	1	Scroll Bar 1	Scroll Bar			☐	☐	
9	1	Checklist Date	Edit Box	CHECKLIST_DT	EMPL_CHECKLIST	☐	☐	
10	1	derived_hr.effdt	Edit Box	EFFDT	DERIVED_HR	☐	☐	
11	1	Checklist	Edit Box	CHECKLIST_CD	EMPL_CHECKLIST	☑	☐	
12	1	Checklist Descript	Edit Box	DESCR	CHECKLIST_TBL	☐	☑	11
13	1	Responsible ID	Edit Box	RESPONSIBLE_ID	EMPL_CHECKLIST	☑	☐	
14	1	Responsible Nam	Edit Box	NAME	PERSONAL_DATA	☐	☑	13
15	1	Comment	Long Edit Box	COMMENTS	EMPL_CHECKLIST	☐	☐	
16	2	Scroll Bar 2	Scroll Bar			☐	☐	
17	2	Chklst Seq	Edit Box	CHECKLIST_SEQ	EMPL_CHKLST_ITM	☐	☐	
18	2	Chklst Itm	Edit Box	CHKLST_ITEM_CD	EMPL_CHKLST_ITM	☑	☐	
19	2	Briefing Descripti	Edit Box	DESCR	CHKLST_ITEM_TBL	☐	☑	18
20	2	Briefing Status	Drop Down List	BRIEFING_STATUS	EMPL_CHKLST_ITM	☐	☐	
21	2	Status Date	Edit Box	STATUS_DT	EMPL_CHKLST_ITM	☐	☐	

EMPLOYEE_CHECKLIST page

EMPL_CHECKLIST is the primary database record for that scrollbar on the EMPLOYEE_CHECKLIST page. The following PeopleCode access the level 1 rowset using EMPL_CHECKLIST. The Business Interlink object name is QE_BI_EMPL_CHECKLIST. This Business Interlink object uses the level 1 rowset as its input and its output.

The **InputRowset** method uses this rowset as input for QE_BI_EMPL_CHECKLIST. Then a blank duplicate of the rowset is created with CreateRowset, and then the output of QE_BI_EMPL_CHECKLIST is fetched into the blank rowset with **FetchIntoRowset**.

```
&MYROWSET = GetRowset(SCROLL.EMPL_CHECKLIST);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.InputRowset(&MYROWSET);
&RSLT = &QE_BI_EMPL_CHECKLIST.Execute();
/* do some error processing */
&WorkRowset = CreateRowset(&MYROWSET);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.FetchIntoRowset(&WorkRowset);

If &ROWCOUNT = 0 Then
   /* do some error processing */
Else
   /* Process the rowset from QE_BI_EMPL_CHECKLIST. Check it for errors. */
   For &I = 1 to &WorkRowset.RowCount
      For &K = 1 to &WorkRowset(&I).RecordCount
         &REC = &WorkRowset(&I).GetRecord(&K);
```

```
            &REC.ExecuteEdits();
            For &M = 1 to &REC.FieldCount
               If &REC.GetField(&M).EditError Then
                  /* there are errors */
                  /* do other processing */
               End-If;
            End-For;
         End-For;
      End-for;
   End-if;
```

Using BIDocs Hierarchical Methods

This section describes the BIDocs methods that are used for hierarchical input and output data.

See Chapter 6, "Business Interlink Class Methods and Properties," Supporting Standard Input/Output with BIDocs Methods, page 38.

AddDoc

Syntax

AddDoc(*docname*)

Description

The **AddDoc** method adds a document to a BIDocs input document. The added document is an input parameter for a Business Interlink object that is not of simple type (such as integer or string). You must add the document to the BIDocs input document before you can add values to its members with **AddValue**.

Parameters

Parameter	Description
Docname	The name of the document that **AddDoc** adds to the BIDocs document.

Returns

The document added to the BIDocs input document.

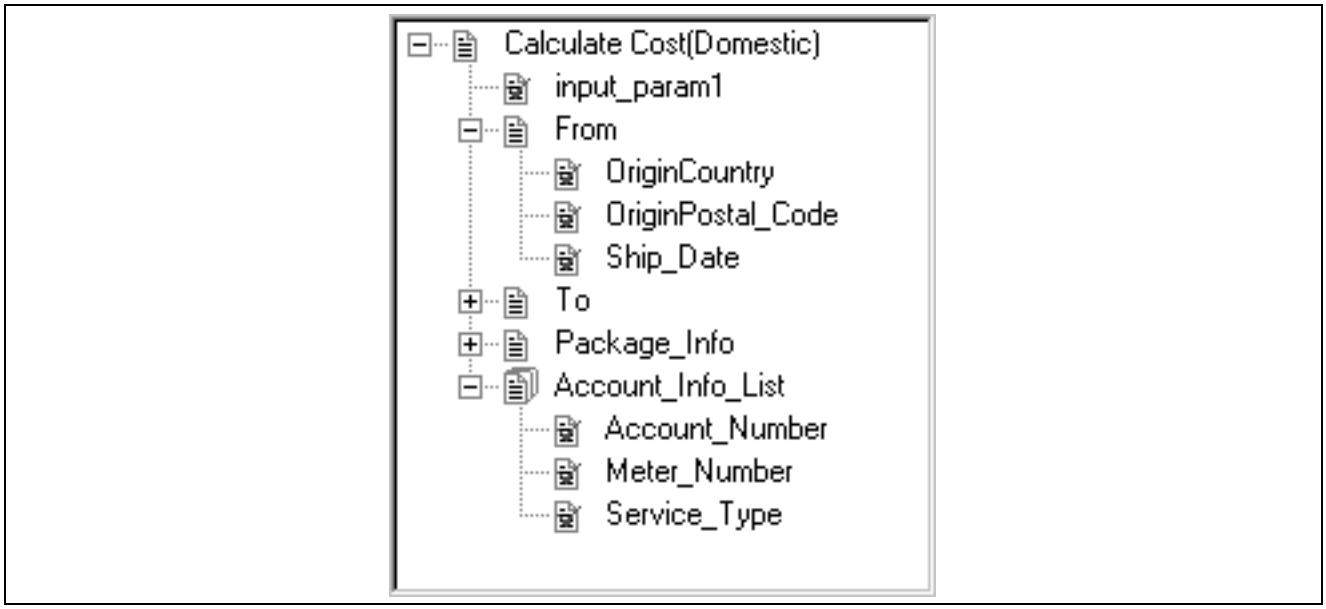
Example

In the following example, the BIDocs input document for Calculate Cost (the root level document) is created with the **GetInputDocs** method. (If you want to create or add more BIDocs input documents, call **AddNextDoc.**) The Calculate Cost input parameter From is a document, so the **AddDoc** method adds it to the input document.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn;
Local BIDocs &FromDoc, &ToDoc, &PackageDoc, &AccountDoc;
Local number &ret, &retinput;
```

```
&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&CalcCostIn = &QE_COST.GetInputDocs("");

&ret = &CalcCostIn.AddValue("input_param1","value");
&FromDoc = &CalcCostIn.AddDoc("From");
&ret = &FromDoc.AddValue("OriginCountry", "United States");
&ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
&ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

/* Call AddDoc and AddValue for To, Package_Info, and Account_Info_
List   (code not shown) */
```

The figure below shows the BIDocs input document for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List.



Example of a BIDocs input document

AddNextDoc

Syntax

AddNextDoc()

Description

The **AddNextDoc** method adds a document to one of the following levels:

- The root level of the BIDocs input document for a Business Interlink object. This level was created with the method **GetInputDocs**.
- When a document within the BIDocs input document is a list, **AddNextDoc** adds another document to the list. If you use **AddNextDoc** on a document that is not a list, **AddNextDoc** fails and returns an error.

Parameters

None.

Returns

A number representing the return status.

Number	Enum value and meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

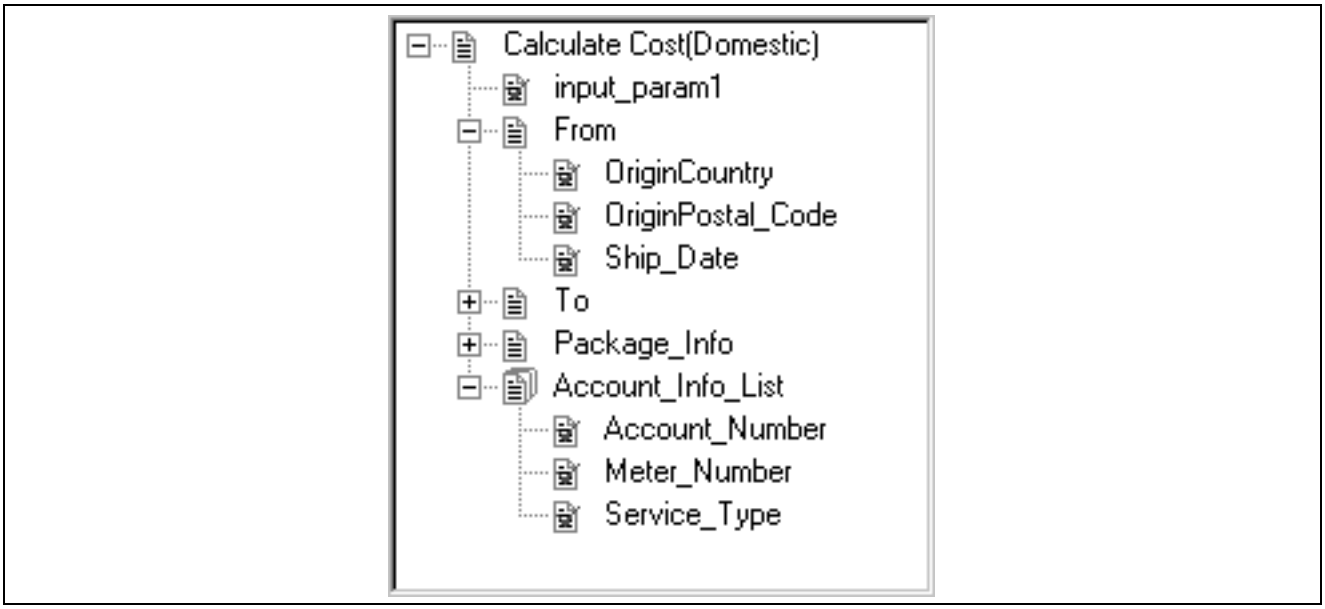
In this example, the BIDocs input document for Calculate Cost (the root level document) is created with the **GetInputDocs** method. The **AddNextDoc** method adds another BIDocs input document. The Calculate Cost input parameter Account_Info_List is a document, so the **AddDoc** method adds it to the BIDocs input document. Account_Info_List is also a document list, so **AddNextDoc** method adds another Account_Info_List document.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn;
Local BIDocs &FromDoc, &ToDoc, &PackageDoc, &AccountDoc;
Local number &ret, &retinput;


&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&CalcCostIn = &QE_COST.GetInputDocs("");

For &n = 1 to &number_of_input_sets
/* Get values for inputs, such as &ORIGIN (code not shown) */

   &ret = &CalcCostIn.AddValue("input_param1","value");
   &FromDoc = &CalcCostIn.AddDoc("From");
   &ret = &FromDoc.AddValue("OriginCountry", "United States");
   &ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
   &ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

   /* Call AddDoc and AddValue for To and Package_Info
      (code not shown) */

   /* Call AddDoc and AddNextDoc for the AccountDoc document list */
   &AccountDoc = &CalcCostIn.AddDoc("Account_Info_List");
   For &m = 1 to &number_of_AccountDocs
      &ret = &AccountDoc.AddValue("Account_Number", &ACCOUNT);
      &ret = &AccountDoc.AddValue("Meter_Number", &METER);
      &ret = &AccountDoc.AddValue("Service_Type", &SVCTYPE);
      &retinput = &AccountDoc.AddNextDoc();
   End-For;

   &retinput = &CalcCostIn.AddNextDoc();

End-For;
```

The figure below shows the BIDocs input document for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List.

From, To, and Package_Info are not lists, so if you try to use **AddNextDoc** with these parameters, such as the following line of code, **AddNextDoc** will fail and return an error message.

```
&retinput = &To.AddNextDoc();
```



Example of a BIDocs input document

AddValue

Syntax

AddValue(*paramname, value*)

Description

The **AddValue** method adds a value to an input parameter within a BIDocs document for a Business Interlink object.

Parameters

Parameter	Description
paramname	The name of the member of the document that is having a value added to it.
value	The value that is added. This can be a constant, a variable, or a record field. The data type can be string, integer, double, float, time, date, or datetime.

Returns

A number representing the return status.

Number	Enum value and meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.

Number	Enum value and meaning
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

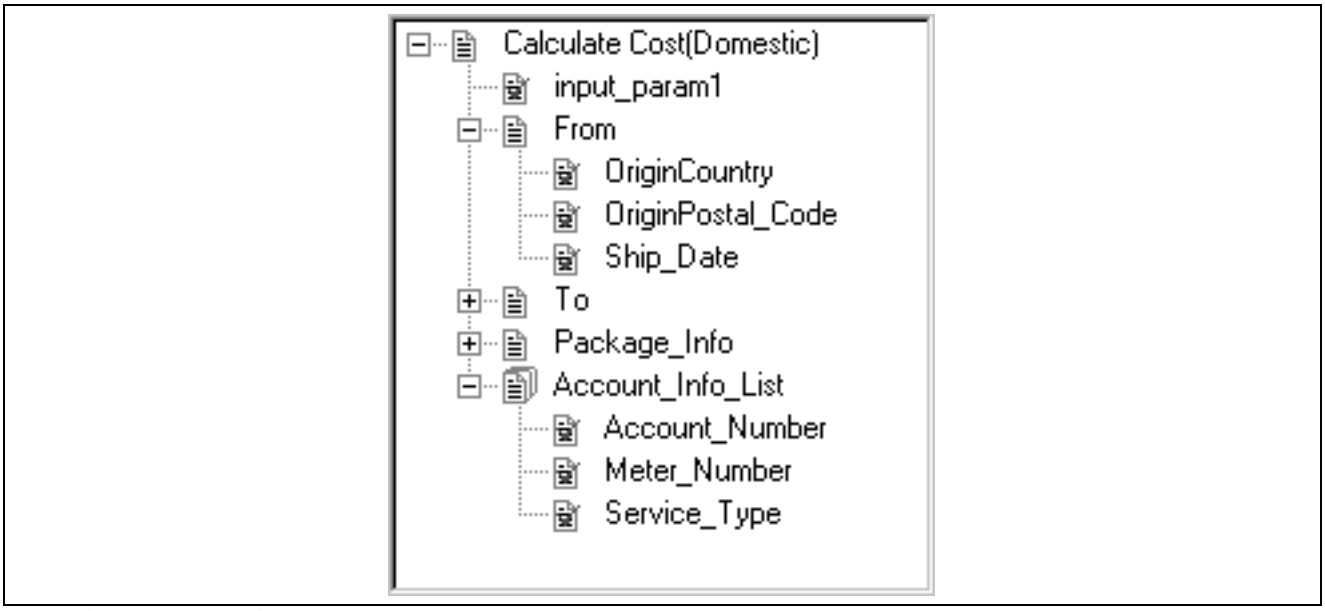
In the following example, the Business Interlink object name is QE_COST.

In the following example, the BIDocs input document for Calculate Cost (the root level document) is created with the **GetInputDocs** method. (If you want to create or add more BIDocs input documents, call **AddNextDoc.**) The Calculate Cost input parameter From is a document, so the **AddDoc** method adds it to the BIDocs input document. Then the **AddValue** method adds values to each of the From document members.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn;
Local BIDocs &FromDoc, &ToDoc, &PackageDoc, &AccountDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

/* Get some values for input, such as &ORIGIN (code not shown) */
```

```
&CalcCostIn = &QE_COST.GetInputDocs("");
&ret = &CalcCostIn.AddValue("input_param1","value");

&FromDoc = &CalcCostIn.AddDoc("From");
&ret = &FromDoc.AddValue("OriginCountry", "United States");
&ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
&ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

/* Call AddDoc, AddValue for Package, Account_Info_List, and also call
AddNextDoc for Account_Info_List (code not shown) */
```

The figure below shows the BIDocs input document for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List.



Example of a BIDocs input document

Clear

Syntax

Clear()

Description

The **Clear** method clears the BIDocs document for a Business Interlink object. If you're using Business Interlinks in a batch program, every time after you use the **Execute** method, you want to use the **Clear** method to flush out the BIDocs document.

Parameters

None.

Returns

None.

Example

In the following example, the BIDocs output document for Calculate Cost (the root level document) is created with the **GetOutputDocs** method. The Calculate Cost output parameter output_param2_List is a document, so the **GetDoc** method gets it from the BIDocs output document. Since output_param2_List is a document list, **GetCount** gets the number of documents in the list.

After the output_param2_List document is processed, the BIDocs document is cleared with **Clear**.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
&count = &CalcCostOut.GetCount("output_param2_List");

&I = 0;
While (&I < &count)
   &ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
   &ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
   If &ret = 0 Then
      /* Process output values */
      &I = &I + 1;
      &retoutput = &OutlistDoc.GetNextDoc();
   End-If;
End-While;

&CalcCostOut.Clear();
```

GetCount

Syntax

GetCount(*docname*)

Description

The **GetCount** method returns the number of documents within a document list or parameter list contained within a BIDocs output document for a Business Interlink object.

Parameters

Parameter	Description
docname	The name of the document list or parameter list.

Returns

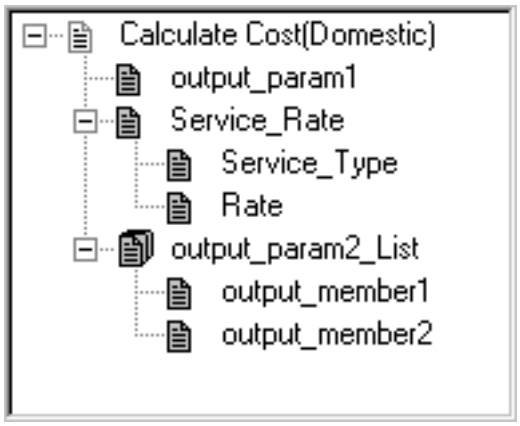
The number of documents in the list.

Example

In the following example, the BIDocs output document for Calculate Cost (the root level document) is created with the **GetOutputDocs** method. The Calculate Cost output parameter output_param2_List is a document, so the **GetDoc** method gets it from the BIDocs output document. Since output_param2_List is a document list, **GetCount** gets the number of documents in the list.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");

/* Call GetValue for output_param1, call GetDoc, GetValue for Service_Rate?
 (code not shown) */

&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
&count = &CalcCostOut.GetCount("output_param2_List");

&I = 0;
While (&I < &count)
   &ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
   &ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
   If &ret = 0 Then
      /* Process output values */
      &I = &I + 1;
      &retoutput = &OutlistDoc.GetNextDoc();
   End-If;
End-While;
```

The figure below shows the BIDocs output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example of a BIDocs output document

GetDoc

Syntax

GetDoc(*docname*)

Description

The **GetDoc** method gets a document from a BIDocs output document. The document is an output parameter for a Business Interlink object that is not of simple type (such as integer or string). You must get the document from the BIDocs output document before you can get values from its members with **GetValue**.

You can call **GetDoc** using a nesting feature. This feature allows you to access deeply nested documents with one call to **GetDoc**, rather than calling **GetDoc** for each nesting. See the Example section below for an example of this.

Parameters

Parameter	Description
docname	The name of the document that **GetDoc** gets from the BIDocs document.

Returns

The document received from the BIDocs output document.

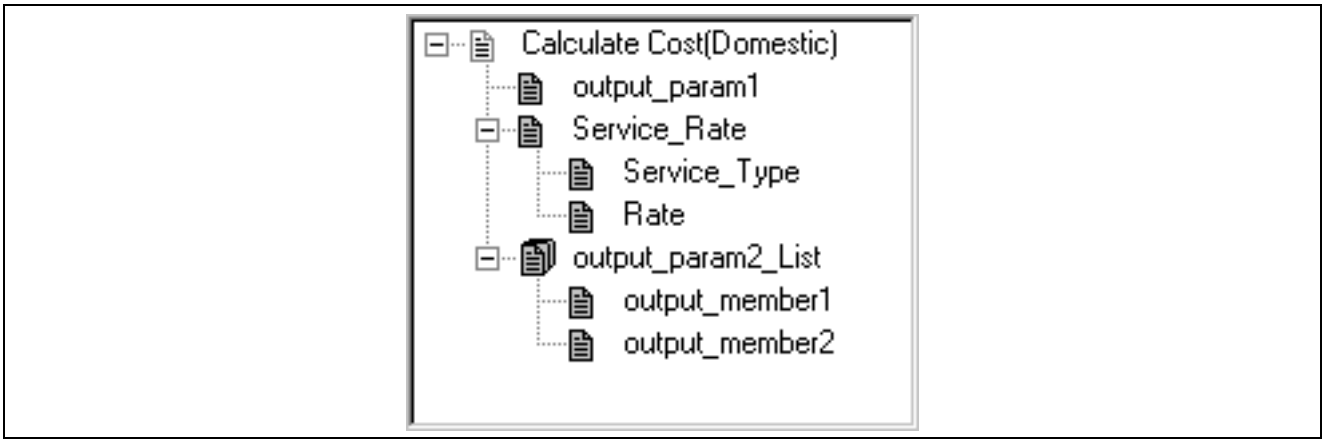
Example

In the following example, the BIDocs output document for Calculate Cost(Domestic), (the root level document) is created with the **GetOutputDocs** method. (If you want to create or get more BIDocs output documents, call **GetNextDoc.**) The Calculate Cost output parameter Service_Rate is a document, so the **GetDoc** method gets it from the BIDocs output document.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &ServiceRateDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
```

```
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&ret = &CalcCostOut.GetValue("output_param1",&VALUE);
&ServiceRateDoc = &CalcCostOut.GetDoc("Service_Rate");
&ret = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
&ret = &ServiceRateDoc.GetValue("Rate", &RATE);
/* Call GetDoc, GetValue, GetNextDoc for output_param2_List (code not shown) */
If &ret = 0 Then
   /* Process output values */
End-If;
```

The figure below shows the BIDocs output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example of a BIDocs output document

In the following example, **GetDoc** is used to access a document that is nested more deeply. If you want to access a document that is more deeply nested, you can either call **GetDoc** for each nesting, or you can call **GetDoc** once using the nesting feature.

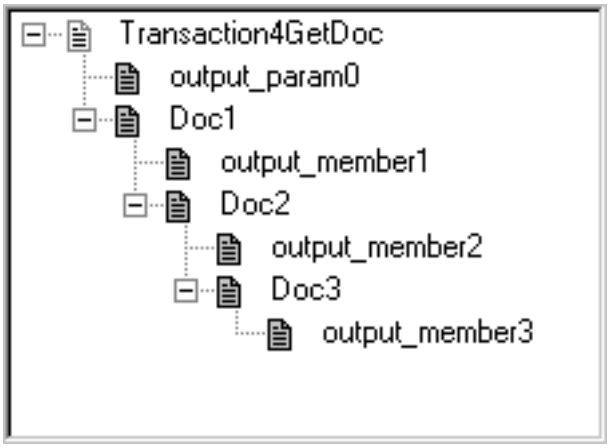
Calling **GetDoc** with the nesting feature:

```
Local Interlink &QE_4GETDOC;
Local BIDocs &CalcCostOut, &Docs3;
Local number &ret;

&QE_4GETDOC = GetInterlink(Interlink.QE_4GETDOC_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_4GETDOC.GetOutputDocs("");
&Docs3 = &CalcCostOut.GetDoc("Doc1.Doc2.Doc3");
&ret = &Docs3.GetValue("output_member3", &VALUE);
```

Calling **GetDoc** without the nesting feature:

```
Local Interlink &QE_4GETDOC;
Local BIDocs &CalcCostOut, &Docs1, &Docs2, &Docs3;
Local number &ret;

&QE_4GETDOC = GetInterlink(Interlink.QE_4GETDOC_EX);
// Get inputs, execute. (code not shown)
```

```
&CalcCostOut = &QE_4GETDOC.GetOutputDocs("");
&Docs1 = &CalcCostOut.GetDoc("Doc1");
&Docs2 = &Docs1.GetDoc("Doc2");
&Docs3 = &Docs2.GetDoc("Doc3");
&ret = &Docs3.GetValue("output_member3", &VALUE);
```



Example of a BIDocs output document: nested documents

GetNextDoc

Syntax

GetNextDoc()

Description

The **GetNextDoc** method gets a document from one of the following levels:

- The root level of the BIDocs output document for a Business Interlink object. This level was created with the method **GetOutputDocs**.
- When a document within the BIDocs output document is a list, **GetNextDoc** gets another document from the list. If you use **GetNextDoc** on a document that is not a list, **GetNextDoc** fails and returns an error.

Parameters

None.

Returns

A number representing the return status.

Number	Enum value and meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.

Number	Enum value and meaning
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

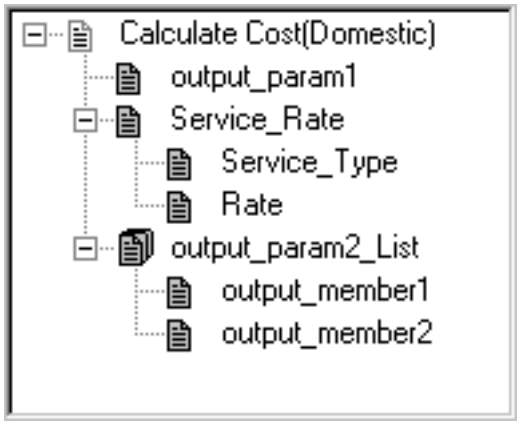
Example

In this example, the BIDocs output document for Calculate Cost (the root level document) is created with the **GetOutputDocs** method. The **GetNextDoc** method gets another BIDocs output document, assuming that there is more than one of them. The Calculate Cost output parameter output_param2_List is a document, so the **GetDoc** method gets it to the BIDocs output document. output_param2_List is also a document list, so **GetNextDoc** method gets the next output_param2_List document.

```
Local Interlink &QE_COST;
Local number &count1, &count2;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret1, &ret2;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)
```

```
&CalcCostOut = &QE_COST.GetOutputDocs("");

&ret1 = 0;
While (&ret1)
   &ret1 = &CalcCostOut.GetValue("output_param1",&VALUE);
   &ServiceRateDoc = &CalcCostOut.GetDoc("Service_Rate");
   &ret1 = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
   &ret1 = &ServiceRateDoc.GetValue("Rate", &RATE);
   /* Process output values */

   &OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
   &count2 = &CalcCostOut.GetCount("output_param2_List");
   While (&I < &count2)
      &ret2 = &OutlistDoc.GetValue("output_member1", &VALUE1);
      &ret2 = &OutlistDoc.GetValue("output_member2", &VALUE2);
      If &ret2 = 0 Then
         /* Process output values */
         &I = &I + 1;
         &ret2 = &OutlistDoc.GetNextDoc();
      End-If;
   &ret1 = &CalcCostOut.GetNextDoc();
End-While;
```

The figure below shows the BIDocs output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example of a BIDocs output document

GetPreviousDoc

Syntax

GetPreviousDoc()

Description

The **GetPreviousDoc** method gets the previous document from either the root level of the BIDocs output document for a Business Interlink object, or from a document within the BIDocs output document. When these documents are in a list, **GetPreviousDoc** gets the previous document in the list. You must get the document before you can get its values with **GetValue**.

Parameters

None.

Returns

A number representing the return status.

Number	Enum value and meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.

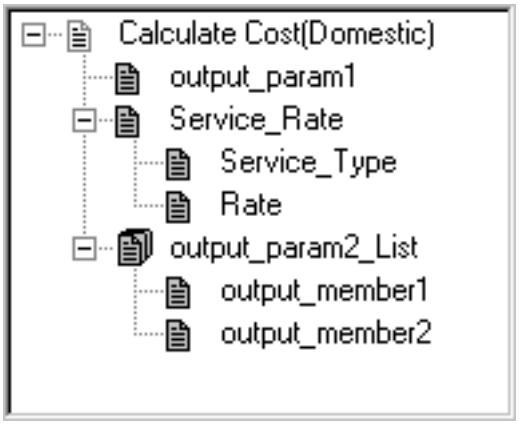
Number	Enum value and meaning
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

In this example, the BIDocs output document for Calculate Cost (the root level document) is created with the **GetOutputDocs** method. It contains one output parameter, output_param2_List, which is also a list. The **GetCount** and **MoveToDoc** methods point to the last output_param2_List document in the list. The **GetPreviousDoc** method is used in a loop to cycle through the output_param2_List list, starting with the last and ending with the first in the list, and get each output_param2_List document and its values.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");

/* Call GetValue for output_param1, call GetDoc, GetValue for Service_Rate?
 (code not shown) */

&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
&count = &CalcCostOut.GetCount("output_param2_List");
&ret = &OutlistDoc.MoveToDoc(&count-1);
&I = &count;
While (&I > 0)
   &ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
   &ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
   If &ret = 0 Then
      /* Process output values */
      &I = &I - 1;
      &retoutput = &OutlistDoc.GetPreviousDoc("");
   End-If;
End-While;
```

The figure below shows the BIDocs output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example of a BIDocs output document

GetStatus

Syntax

`GetStatus``()`

Description

The **GetStatus** method tests the BIDocs document. To find the status for any BIDocs method that does not return a status value, call GetStatus just after you call that BIDocs method.

Parameters

None.

Returns

A number representing the return status.

Number	Enum value and meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.

Number	Enum value and meaning
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

GetValue

Syntax

```
GetValue(paramname, value)
Number GetValue(paramname, value)
```

Description

The **GetValue** method gets a value from an output parameter within a BIDocs output document for a Business Interlink object.

Parameters

Parameter	Description
paramname	The name of the member of the document that is having a value retrieved from it.
value	The value that is retrieved. This can be a variable or a record field. The data type can be string, integer, double, float, time, date, or datetime.

Returns

A number representing the return status.

Value	Meaning
string, *value*	The value of the output parameter.
Number	When the syntax is Number GetValue(*name*, *value*) number is the return status of GetValue, listed in the table below.

The following table contains the return status values when the syntax is Number GetValue(*name*, *value*).

Number	Enum value and meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.

Number	Enum value and meaning
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

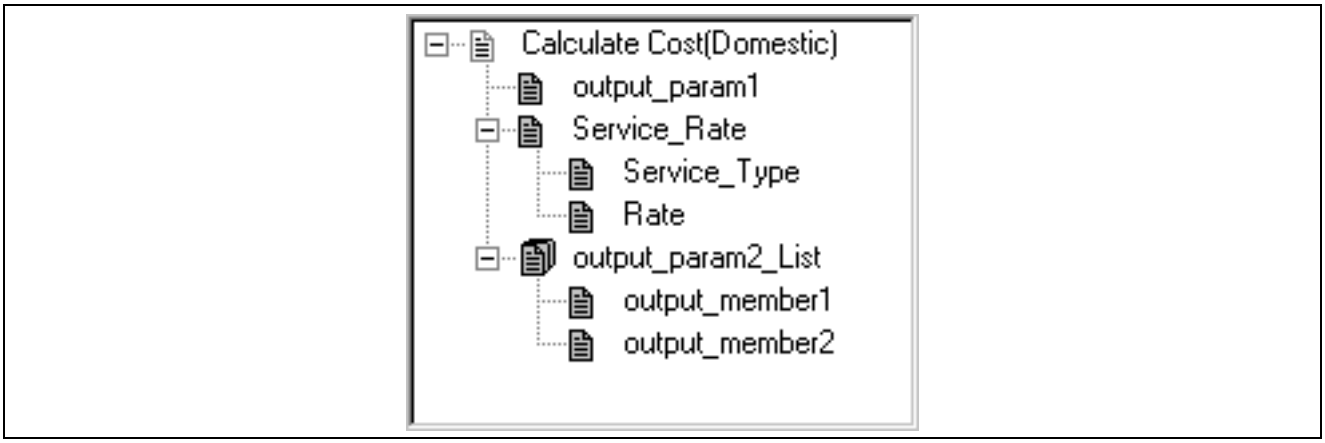
Example

In the following example, the Business Interlink object name is QE_COST.

In the following example, the BIDocs output document for Calculate Cost (the root level document) is created with the **GetOutputDocs** method. (If you want to create or get more BIDocs output documents,call **GetNextDoc.**) The Calculate Cost output parameter Service_Rate is a document, so the **GetDoc** method gets it from the BIDocs output document. Then the **GetValue** method gets values from each of the Service_Rate document members.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostOut;
Local BIDocs &ServiceRateDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&ret = &CalcCostOut.GetValue("output_param1",&PARAM1);

&ServiceRateDoc = &CalcCostOut.GetDoc("Service_Rate");
&ret = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
&ret = &ServiceRateDoc.GetValue("Rate", &RATE);

/* Call GetDoc, GetValue, GetNextDoc for output_param2_List (code not shown) */
```

The figure below shows the BIDocs output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example of a BIDocs output document

MoveToDoc

Syntax

MoveToDoc(*list_number*)

Description

Within a list of documents in the BIDocs output document, the **MoveToDoc** method moves to the documents given by the parameter *list_number*. After using **MoveToDoc**, the **GetValue** method gets the values of the document that is in the *list_number*+1 location in the list.

Parameters

Parameter	Description
list_number	The number indicating the document that **MoveToDoc** moves to. After using **MoveToDoc**, the **GetValue** method gets the values of the document that is in the *list_number*+1 location in the list. For example, if *list_number* is zero, then **MoveToDoc** moves to the first document in the list.

Returns

A number representing the return status.

Number	Enum value and meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.

Number	Enum value and meaning
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

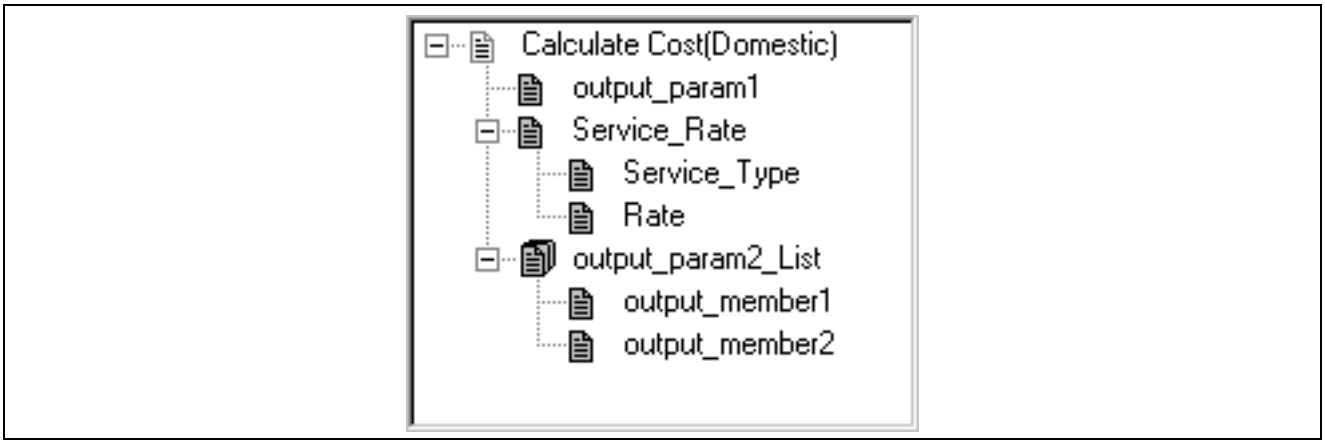
Example

The following example gets the values of the last document in the output_param2_List list. It uses **GetCount** to get the number of documents in the list, and then uses **MoveToDoc** to move to the last document in the list.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
```

```
&count = &CalcCostOut.GetCount("output_param2_List");
&ret = &OutlistDoc.MoveToDoc(&count-1);
&ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
&ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
```

The figure below shows the BIDocs output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example of a BIDocs output document

ResetCursor

Syntax

ResetCursor()

Description

The **ResetCursor** method resets the cursor in the BIDocs output document for a Business Interlink object to the top. Once you call this method, the next time you call **GetValue**, you get an output value from the first document of the BIDocs output documents for a Business Interlink object.

Parameters

None.

Returns

None.

Example

The following code uses **ResetCursor** to reset the cursor in the BIDocs output document to the top.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostOut;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
```

```
// Perform actions on the output documents (code not shown)

&CalcCostOut.ResetCursor();
```

Using the Interlink Property

In this section, we discuss the Interlink property.

StopAtError

Description

Specifies whether the execution of the PeopleCode program will stop when there's an error, or if the PeopleCode program will try to capture the errors. This property takes a Boolean value: True to halt execution of your PeopleCode program at an error, False to continue executing. The default value is True.

This property is read-write.

Example

```
&QE_RP_SRAALL_1.StopAtError = False;
```

Using Configuration Parameters

This section provides an overview of configuration parameters and discusses how to:

- Use the Business Interlink plug-in configuration parameters.
- Use the URL configuration parameter.
- Use the BIDocValidate configuration parameter.

Understanding Configuration Parameters

There are two types of configuration parameters: the ones defined by the Business Interlink plug-in the Business Interlink definition is based on, and ones that are standard, that is, defined for every Business Interlink object.

All configuration parameters must be set before you add any data to the input buffers.

See Chapter 4, "Generating the PeopleCode Template," page 23.

Using Business Interlink Plug-in Configuration Parameters

The configuration parameters defined by the Business Interlink plug-in are accessed as if they were properties on the Business Interlink object. That is, in PeopleCode, you assign the value of a configuration parameter by using the Business Interlink object followed by a dot and the configuration parameter, in this format:

```
&MYINTERLINK.parameter = value;
```

You can also return the value of a configuration parameter:

```
&MYVALUE = &MYINTERLINK.parameter;
```

Each Business Interlink plug-in has its own set of configuration parameters. You can specify default values for every configuration parameter in the Business Interlink definition (created in the Application Designer.) These values will be used if you create a PeopleCode template by dragging the Business Interlink definition from the Project window in the Application Designer to an open PeopleCode editor window.

In the following example, the Business Interlink object name is QE_RP_SRAALL_1:

```
&QE_RP_SRAALL_1.SERVER_NAME = "server";
&QE_RP_SRAALL_1.RSERVER_HOST = "pt-sun02.peoplesoft.com";
&QE_RP_SRAALL_1.RSERVER_PORT = "9884";
&QE_RP_SRAALL_1.TIMEOUT = 999;
&QE_RP_SRAALL_1.URL = "HTTP://www.PeopleSoft.com";
&QE_RP_SRAALL_1.StopAtError = False;
```

Using the URL Configuration Parameter

Specifies the location and name of the Business Interlink plug-in to be used to connect to the external system. This configuration parameter takes a string value.

If the plug-in is located on a web server, you have to specify the name of the web server.

If you specify a file, you can specify either a relative or an absolute path:

- If you specify an absolute path, you must specify the drive letter:

```
&MYBI.URL = File://D:\TEMP.MYDLL;
```

- If you specify a relative path, just use the file name:

```
&MYBI.URL = File://TEMP.MYDLL;
```

If you specify a relative path, the system will first look for the file in the Location directory (specified by the user when the Business Interlink was first created), then it will look in the directory where PeopleTools is installed, in the PSTOOLS/Interface Drivers directory.

Using the BIDocValidate Configuration Parameter

When set to ON, a BIDocs object is checked to see if it exists before adding/getting values from it. For example, if the BIDoc object “Rate” does not exist, and BIDocValidate is set to ON, then the following line of code will cause an error:

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

Using the Business Interlink Function

This section lists the Business Interlink function.

GetInterlink

Syntax

```
GetInterlink(Interlink.interlinkname)
```

Description

The **GetInterlink** function instantiates a Business Interlink Object based on a Business Interlink created in the Application Designer. This function is part of the PeopleSoft Business Interlink framework, which can provide a gateway for PeopleSoft applications to the services of any external system.

Parameters

Parameter	Description
Interlink.interlinkname	Specify the name of the business interlink (created in the Application Designer) from which to instantiate a Business Interlink Object.

Returns

A Business Interlink Object.

Example

The following example instantiates a Business Interlink Object based on the Business Interlink Definition QE_RP_SRAALL.

```
Local Interlink &SRA_ALL_1;
&SRA_ALL_1 = GetInterlink(Interlink.QE_RP_SRAALL);
```

PART 2

Business Interlinks for Design-Time Plug-In Programming

CHAPTER 7

Designing Business Interlink Transactions and Classes

This chapter provides an overview of Business Interlink transactions and classes and discusses how to:

- List your configuration parameters.
- List your business classes.
- List your business transactions.

Understanding Business Transactions and Classes

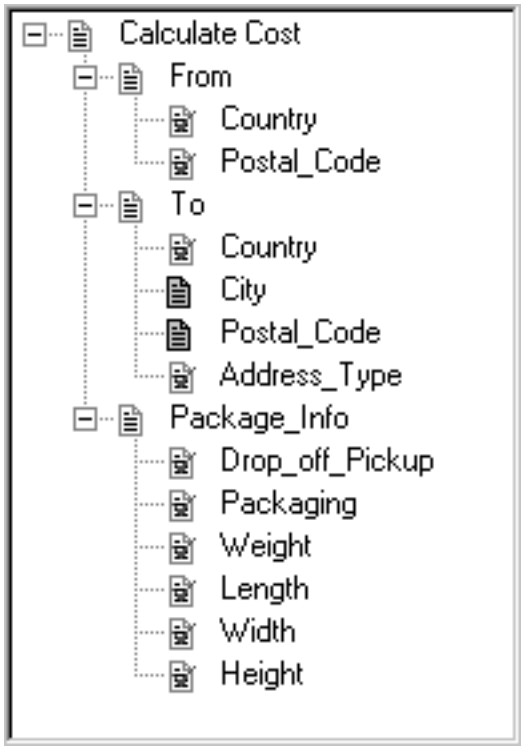
When you use Business Interlinks with your system, you must design a set of Business Interlink transactions and/or classes. A *business transaction*, or transaction, is a named command with optional named and typed inputs and outputs. A *business class*, or class, is a definition of a data structure. 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, and 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 of 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.

Here is an example of the Calculate Cost(Domestic) transaction inputs resulting from the designs in this chapter.



Example of the Calculate Cost(Domestic) transaction inputs

Here is an example of the Calculate Cost(Domestic) transaction outputs resulting from the designs in this chapter.

Example of the Calculate Cost(Domestic) transaction outputs

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 (SMTP 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.

If you use configuration parameters other that 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 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 runtime 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.

Data Member Name	Data Type
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, Letter Envelop, Express Box, Worldwide 25KG Box, Worldwide 10KG Box
Weight	Integer
Length	Integer

Data Member Name	Data Type
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
Guaranteed_By	String
Rate	String

Listing 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 provides 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.

Input Parameter Name	Data Type
From	Origin class
To	Destination class
Package_Info	Package_Information class

Output Parameter Name	Data Type
Service_Rate	Service Rate class

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

Input Parameter Name	Data Type
From	Origin class
To	Destination class

Output Parameter Name	Data Type
From	Origin class
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

CHAPTER 8

Writing an XML Design-time Plug-in

This chapter provides an overview of XML design-time plug-ins and discusses how to:

- Create an XML design-time plug-in.
- Write the tags in an XML design-time plug-in.
- Use data types.

Understanding XML Design-time Plug-ins

Once you have designed your transactions and classes by naming the transactions and 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.

Note. 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 that describes how to write design time functionality when you do not write an XML design-time plug-in.

Creating an 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\PSINTERLINKS\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\PSINTERLINKS\SRC\C++\SAMPLES
```

This is an example of 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.

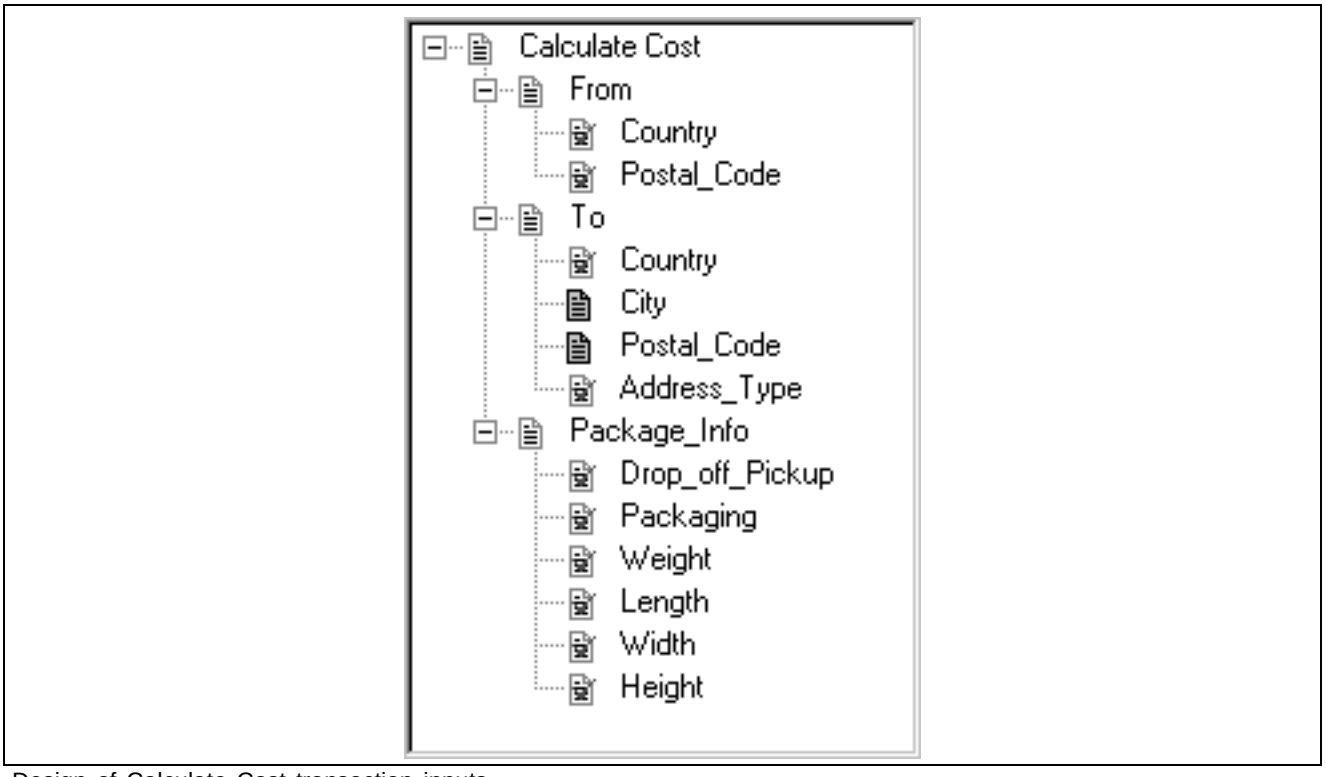
```
<?xml version="1.0" ?>
<interface_driver>

   <general_info>
      <description>PSCustomer services</description>
      <version>1</version>
      <comments>PSCustomer plug-in</comments>
      <image>PSCustomer.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>


   <config_parameters>
      <URL>file://PSCustomer.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,?
Letter Envelop,Tube,Express Box,?
Worldwide 25KG Box,Worldwide 10KG Box )"
                  default="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="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="PSCustomer 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>
```

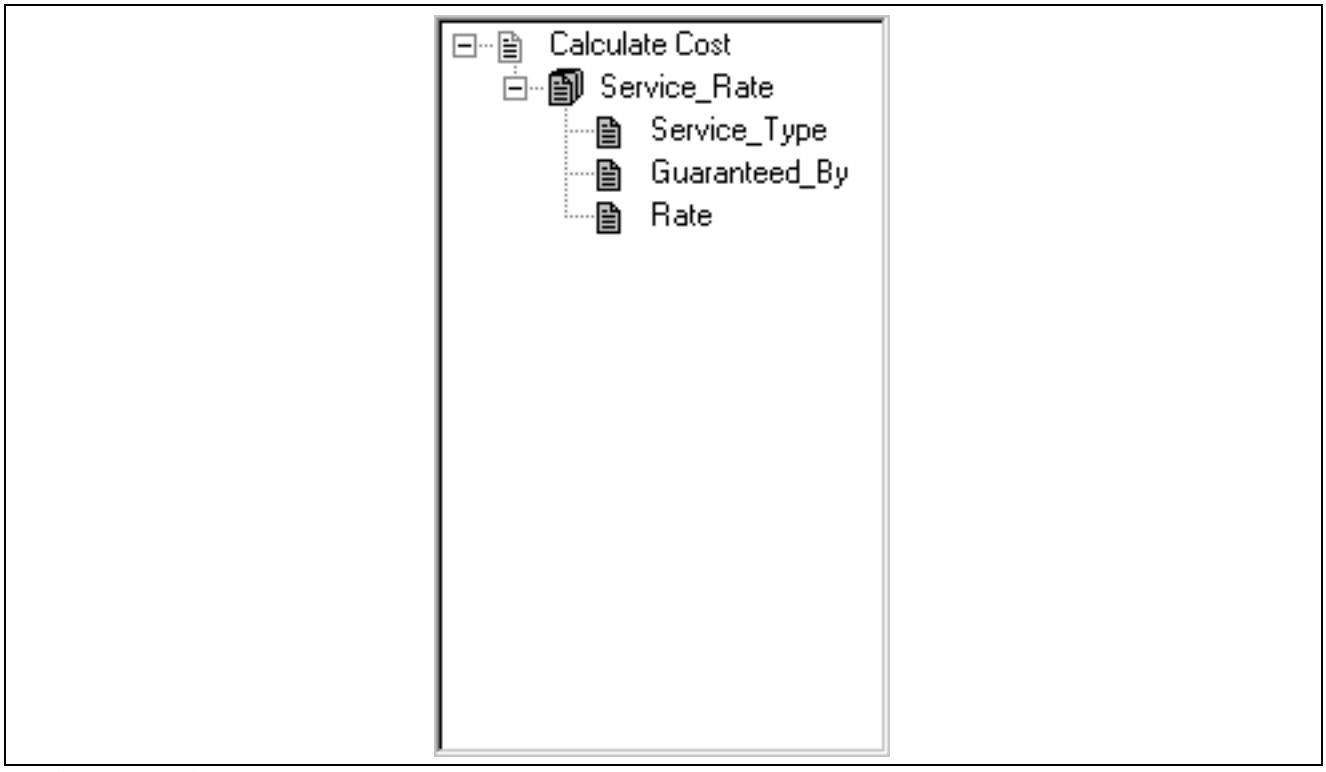
```
                <input name="From"
                   type="object"
                   classname="Origin"/>
                <input name="To"
                   type="object"
                   classname="Destination"/>
                <input name="Package_Info"
                   type="object"
                   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>
```

Here is an example of the Calculate Cost transaction inputs resulting from the above XML design-time plug-in.



Design of Calculate Cost transaction inputs

Here is an example of the Calculate Cost transaction outputs resulting from the above XML design-time plug-in.



Design of Calculate Cost transaction outputs

Writing The Tags in an XML Design-time Plug-in

This section discusses how to:

- Write the main tag <interface driver>.
- List the supported Business Interlink types <driver_settings>.
- List the configuration parameters <config_parameters>.
- Write the class catalog <class_catalog>.
- Write the transaction catalog <trans_catalog>.

Writing the Main Tag <interface_driver>

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.

Writing the General Info Tag <general_info>

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 *PSCustomer services*, and the version number is *1*.

```
<general_info>
   <description>PSCustomer services</description>
   <version>1</version>
   <comments>PSCustomer plug-in</comments>
   <image>PSCustomer.bmp</image>
</general_info>
```

This section discusses how to:

- Write the plug-in description <description>.
- Write the plug-in version number <version>.
- Name a graphic file for the Business Interlink property dialog box <image>.

Write the Plug-in Description <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 *PSCustomer services.*

```
<description>PSCustomer 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.

Write the Plug-in Version Number <version>

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.

See Chapter 11, “Writing the Version Methods for a Business Interlink Runtime Plug-In,” Writing GetVersion for Your Business Interlink Plug-in Class, page 135.

See Chapter 11, “Writing the Version Methods for a Business Interlink Runtime Plug-In,” Writing IsVersionCompatible for Your Business Interlink Plug-in Class, page 135.

Name a Graphic File for the Business Interlink Property Dialog Box <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>PSCustomer.bmp</image>
```

Listing The Supported Business Interlink Types <driver_settings>

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"/>
</driver_settings>
```

Write a Business Interlink Type <option>

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.

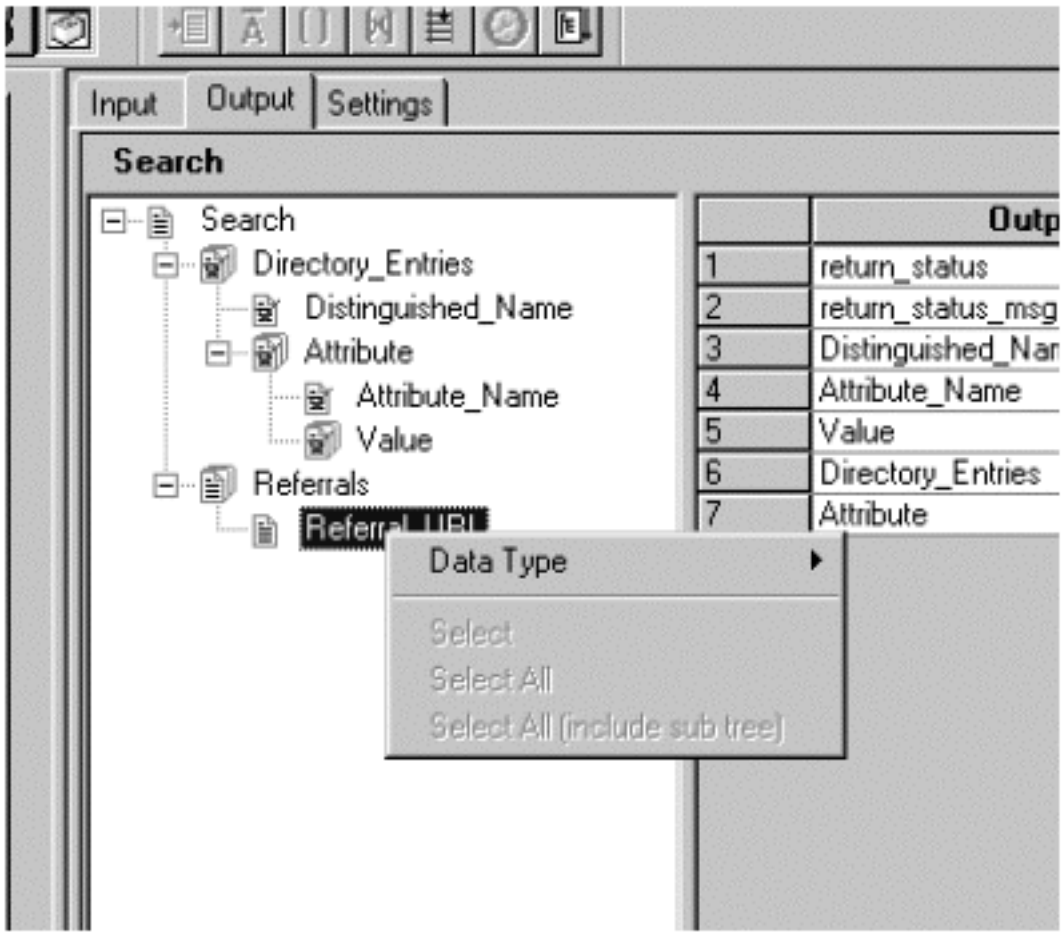
Type	Definition
order_by_desc	If you use the object_query type, you can use order_by_desc for descending order. This type allows a query Business Interlink object to list query results in descending order. The type order_by_desc is only used with query Business Interlink types.
order_by_asc	If you use 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. `<option type="input_class_expandable" ? supported="true"/>`
output_class_expandable	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. `<option type="output_class_expandable"? supported="true"/>`
hierarchical_model	To support Business Interlink object lists, set the hierarchical_model object type.

To support Business Interlink object lists, you should support the hierarchical_model object type in the <driver_settings> section of your XML design-time plug-in:

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

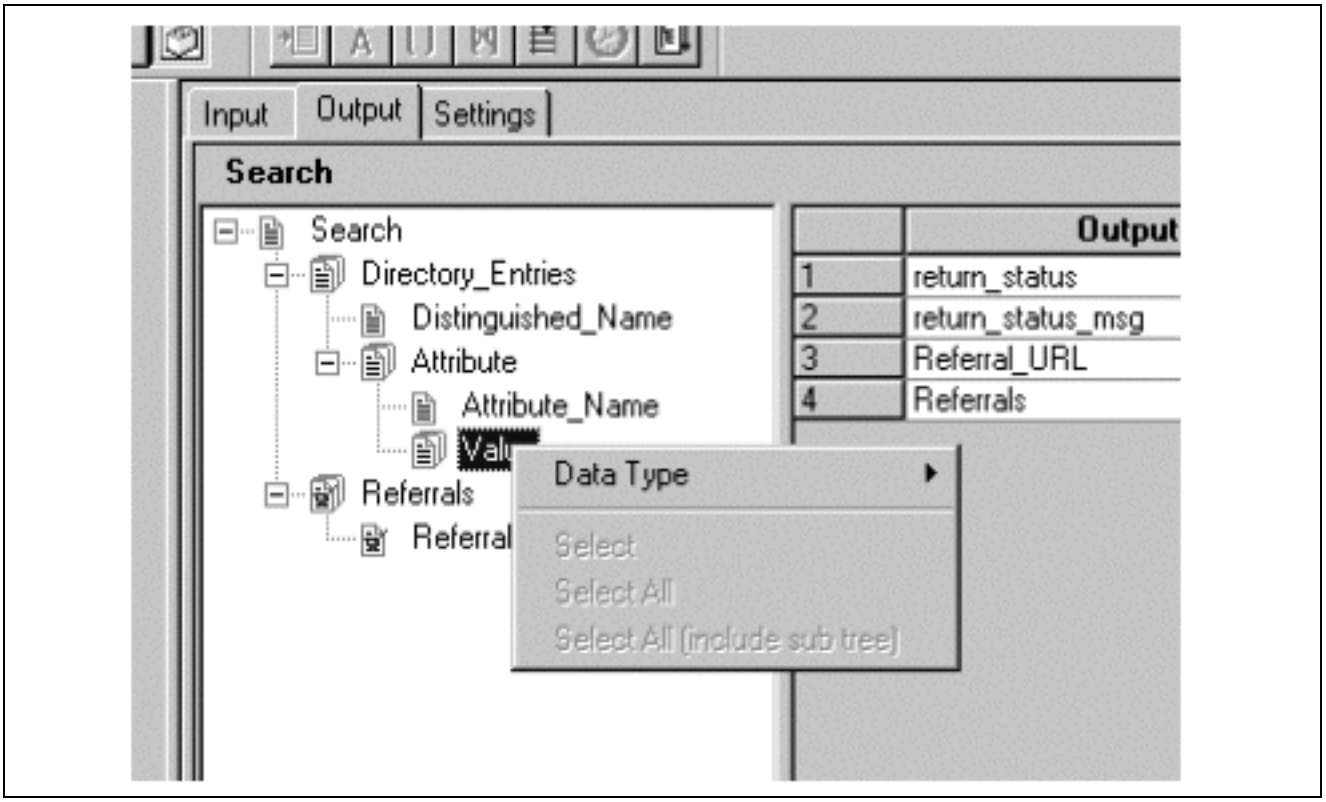
Within the Business Interlink definition created from this XML design-time plug-in, this setting allows you to select the children of two object lists that are at the same level of a Business Interlink definition hierarchy.

For example, the following Business Interlink definition contains object lists at the same level.



Referrals object list not selected

Once a child of Directory_Entries is selected, Referral_URL can not be selected unless the XML design-time plug-in supports the hierarchical_model object type.



Directory_Entries not selected

Likewise, if Referral_URL is selected first, Directory_Entries can not be selected unless the XML design-time plug-in supports the hierarchical_model object type.

Listing The Configuration Parameters <config_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://PSCustomer.dll</URL>
</config_parameters>
```

Note. Whenever you specify a UNIX/Linux 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>
```

This section discusses how to:

- Write a configuration parameter <parameter>.
- Error check Business Interlink documents <BiDocValidate>.
- Specify the Business Interlink runtime plug-in file location <URL>.

Writing a Configuration Parameter <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"/>
```

See Chapter 8, “Writing an XML Design-time Plug-in,” Using Data Types, page 119.

Error Check Business Interlink Documents <BiDocValidate>

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 PeopleCode to access the input and output data for Business Interlinks.

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.

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

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 PSCustomer.dll.

```
<URL>file://PSCustomer.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
```

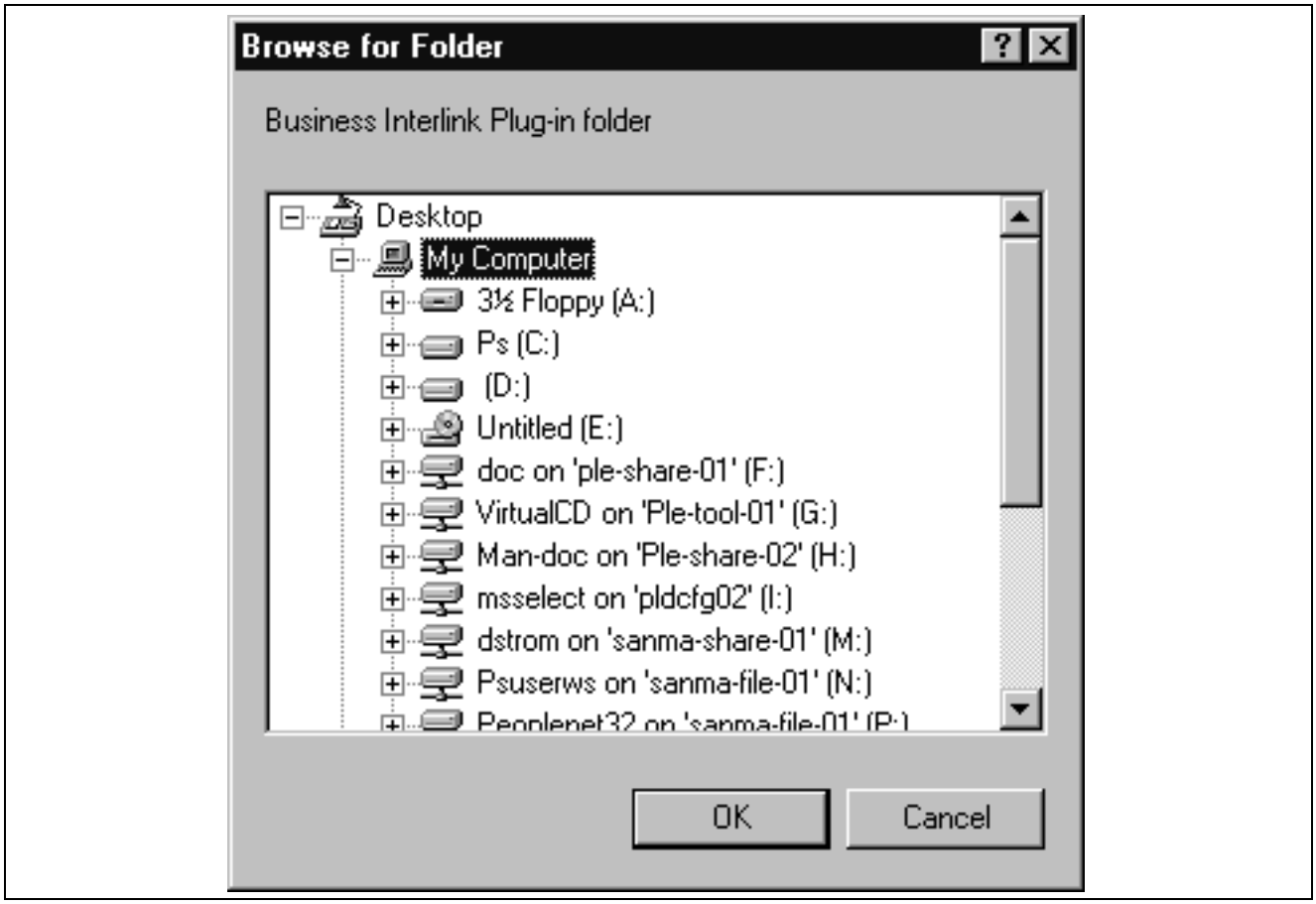
For *servletname* to work, the runtime plug-in must use the same name as the name of the XML design-time plug-in. For example, if runtime is fcarrier.dll, the design-time must be fcarrier.xml.

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 clicking the Location button in the New Business Interlink 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.



Browse For Folder page

Writing the Class Catalog <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 -->

   </category>
</class_catalog>
```

This section discusses how to:

- Write the categories for the classes.
- Write a class.
- Write members for a class.

Write The Categories For The Classes <category>

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

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 -->

</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 -->

</category>
```

Note. 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.

Write A Class <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>
```

Write Members For A Class <member>

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. It can also contain a trim, which causes all leading and trailing spaces to be trimmed from the data member; any spaces within, such as in “value 1”, are not trimmed.

Note. Whenever you specify a UNIX/Linux 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 a data member named *OriginPostal_Code.*

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

When you use trim, and you are building XML tags, you might need to add a space when calling AddValue in your PeopleCode program to control the output. For example, for the following class:

```
<class name="EndDate">
 <member name="Date" type="date" required="false"/>
 <member name="CurrentFlag" type="string" required="false" trim="true"/>
 <member name="SummaryText" type="object" classname="SummaryText" required="false"/>
</class>
```

Within PeopleCode, you can add a space with the member that has the trim:

```
&date = "abc";
&EndDateDoc.AddNextDoc();
If &date <> "" Then
   &ret = &EndDateDoc.AddValue("Date", "2001-8-10");
Else
   &ret = &EndDateDoc.AddValue("CurrentFlag", " ");
End-If;
```

Without the space, you get the following tag:

```
<CurrentFlag/>
```

With the space, you get the following tag:

```
<CurrentFlag> <CurrentFlag/>
```

See Chapter 8, “Writing an XML Design-time Plug-in,” Using Data Types, page 119.

Writing the Transaction Catalog <trans_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 -->

      </category>
   </trans_catalog>
```

This section discusses how to:

- Write the categories for the transactions <category>.
- Write a transaction tag <transaction>.
- List the input parameters <input_list>.
- Write an input parameter <input>.
- List the output parameters <output_list>.
- Write an output parameter <output>.

Write The Categories For The Transactions <category>

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_catalog>** tag or a **<class_catalog>** tag.

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

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

   <!-- Write the transaction tags here -->

</category>
```

Write A Transaction <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.

Note. Whenever you specify a UNIX/Linux 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"/>
```

See Chapter 8, “Writing an XML Design-time Plug-in,” Using Data Types, page 119.

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. An <**output**> tag contains a name and type, and an optional default value. It can also contain a trim, which causes all leading and trailing spaces to be trimmed from the data member; any spaces within, such as in “value 1”, are not trimmed.

Note. Whenever you specify a UNIX/Linux 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" />
```

See Chapter 8, “Writing an XML Design-time Plug-in,” Using Data Types, page 119.

See Chapter 8, “Writing an XML Design-time Plug-in,” Writing the Class Catalog <class_catalog>, page 114.

Using 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	An enumeration. The format is enum(*name1*, *name2*,…) where *name1*, *name2*, … are the comma delimited names for this enumeration. Following is an example of an enumeration named Address with the values of Commercial and Residential. `<member name="Address"? type="enum(Commercial, Residential)">`
Object	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. `<input name="From" type="object"? classname="Origin"/>`

Type	Definition and format
Password	A password. This is a character string.
Lists of the above types (not used with configuration parameters)	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. List types can not be used with relational or logical operators. 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.

Escaping XML Restricted Characters

Anywhere you pass in text or data in the XML design-time plug-in, all XML restricted characters (such as < and >) need to be escaped. For example, < must be %60, > must be%62, and " must be %34.

CHAPTER 9

Deploying an 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.

PART 3

Business Interlinks for Runtime Plug-in Programming

Chapter 18
Configuring PSINTERLINKS as a WebApp for Distributed Business Interlinks

CHAPTER 10

Setting Up a Business Interlink Runtime Plug-in

This chapter discusses how to:

- Set up your development environment for writing a Business Interlink runtime plug-in in C++.
- Set up your development environment for writing a Business Interlink runtime plug-in in Visual Basic.
- Set up your development environment for writing a Business Interlink runtime plug-in in Java.

Setting Up the Development Environment in C++

This section discusses how to:

- Set up the development environment for writing a Business Interlink runtime plug-in in C++ under Windows NT.
- Set up the development environment for writing a Business Interlink runtime plug-in in C++ under UNIX/Linux.
- Create a C++ template.

Setting Up the Development Environment in C++ Under Windows NT

This section discusses how to create the C++ project that you use to write your Business Interlink plug-in under Windows NT.

To create the C++ project that you use to write your Business Interlink plug-in under Windows NT:

1. Within Visual C++, select File, New, or press Ctrl N, and then click the Projects tab.

   The New dialog box, Projects tab appears.

2. Fill out the New Project dialog box, Projects tab as follows, and then click OK.

   In the list view control on the left side, select the Win32 Dynamic-Link Library type.

   In the project name type the name of the project.

   In the location edit boxes, type the location where you want to store your project's files.

3. Select Build: Configurations. The Configurations dialog box appears.
4. Click the Add button. The Add Project Configuration dialog box appears.

   The Configurations dialog box shows all of the different configurations that have been defined for your project. By default, only the Win32 Release and Win32 Debug configurations are defined.

5. Fill out the Add Project Configuration dialog box as follows and then click OK:

a. In the Configuration edit box, change the text from “Debug” to “UnicodeDebug”.

   This is set because, by default, the project is not Unicode enabled, so Unicode must be enabled here.

b. In the “Copy settings from” drop down box, select the Win32 Debug configuration as the template.

6. Repeat steps 4 and 5, except in step 5:

   a. In the Configuration edit box, change the text to “Unicode Release”.

   b. In the “Copy settings from” drop down box, select the Win32 Release configuration as the template.

   c. Click the Close button.

7. Select Build:Set Active Configuration.

   The Set Active Project Configuration dialog box appears.

8. Select Win32 UnicodeDebug and click OK.

9. Select Project: Settings.

   The Project Setting dialog box appears.

   The Settings For drop down box contains a list of each of the configurations that you have defined for your project. This example uses the Win32 UnicodeDebug configuration; the tasks are similar for the other configurations.

10. Click the C/C++ tab, and with the Category drop down box showing General, within the Preprocessor definitions edit box, check to see that the following text exists in the Preprocessor definitions list, and add it if it is not there:

    ```
    UNICODE, _UNICODE, PS_WIN32
    ```

    This will enable Unicode string support during compilation.

11. From the Category drop down box, choose the Preprocessor category, and in the Additional include directories edit box, type in the following path.

    ```
    <PS_HOME>\SDK\PSINTELINKS\src\C++\Inc
    ```

    This is the directory that the compiler will search for the standard PeopleSoft Business Interlink Software Development Kit header files.

12. Click on the Link tab, and with the Category drop down box showing General, edit the Output file name edit box and the Object/library modules edit box as follows:

    Within the Output file name edit box, enter the following path and the name of the output file for your project:

    ```
    <PS_HOME>\bin\client\winx86\InterfaceDrivers\output.dll
    ```

    where *output*.dll is the output file. You should use the same name for this output file as the name of the XML design-time plug-in. For example, if the design-time is fcarrier.xml, the runtime should be fcarrier.dll.

    See Chapter 8, “Writing an XML Design-time Plug-in,” page 101.

    Within the Object/library modules edit box, add the filename psiobase.lib to the end of the list. This will link in the psiobase.lib library with your project during the build process.

13. From the Category drop down box, select Input, and in the Additional library path edit box, type in the following path:

    ```
    <PS_HOME>\SDK\PSINTERLINKS\Lib\
    ```

    This is the path for the psiobase.lib file directory.

14. Click OK to accept the settings and then save your project.

    Your project is now configured. You can start coding.

    You can save the project under the following directory, in order to keep it near the C++ Business Interlink files.

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

    where *project_name* is the name of your project.

15. Copy the file psiodrvr.cpp and edit that copy to create your own C++ runtime plug-in.

    The psiodrvr.cpp file contains most of the structure that you need for your runtime file. Copy the psiodrvr.cpp file from the following directory.

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

Setting Up the Development Environment in C++ Under UNIX/Linux

To write a Business Interlink runtime plug-in using C++, you need to set up the development environment on the UNIX/Linux system.

To set up for your Business Interlink Plug-in in UNIX:

1. Navigate to the <PS_HOME> directory and run the following command:

   ```
   . ./psconfig.sh
   ```

2. Test the setup by running the Business Interlink tester on the executable files in the simple directory.
3. Create the directory for your plug-in.

   You can copy files from the sample directory to your plug-in directory.

   ```
   cd $PS_HOME/sdk/psinterlinks/src/c++/samples
   cp -rf simple yourplugindirname
   ```

4. Edit the makefile in your plug-in directory.

   Replace *simple* with *yourplugindir.*

   ```
   name = yourplugindir
   ```

   For AIX, replace *libext=so* with *libext=aix*

   For HP, replace *libext=so* with “*libext=sl*
5. Create your Business Interlink runtime plug-in.
6. Store the .cpp and .h files within the directory you created.

   You can copy, rename, and use the .cpp and .h files contained in the simple directory as a template.
7. To compile and link your runtime plug-in, run the following make commands:

   ```
   yourplugindirname\unix>make rulesfile
   yourplugindirname\unix>make
   ```

Creating a C++ Template

You can use the following code as a template for the header of your Business Interlink class. In this code, perform the following tasks:

- Include the InterfaceObject class.
- Include the input/output table classes.
- Declare your class so it inherits the DLLBaseDriver class.
- Create a constructor for your class.
- Declare the pure virtual methods GetVersion, IsVersionCompatible, and the execution method you will use:: ExecuteTransaction, ExecuteQuery, ExecuteAdd, ExecuteUpdate, and/or ExecuteDelete. In most cases, you will use ExecuteTransaction.

```
// The include files.
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "iodrvr.h"   // Contains the InterfaceObject class
#include "ioutil.h"   // Contains the input/output table classes
// Also include any classes you need to use your external system.
// Declare your class, and inherit the DLLBaseDriver class.
class yourplugin : public DLLBaseDriver
{
public:
   // Constructor for your class.
yourplugin() : IntObj(0) {}

   // Declare the virtual methods
   virtual const TCHAR* GetVersion() const;
   virtual BOOL  IsVersionCompatible(const TCHAR* version);
   virtual EIOCEXECSTATUS
     ExecuteTransaction(InterfaceObject * IntObj,
     const int nBatchMode);
// Create the InterfaceObject pointer.
public:
   InterfaceObject* IntObj;
};
```

Setting Up the Development Environment in Visual Basic

This section discusses how to:

- Register DLL files for Visual Basic.
- Set up Visual Basic application development.

Registering DLL files for Visual Basic

In order to write your own plug-in in Microsoft Visual Basic, you must have access to the following files:

- IoCollection.DLL
- PsIntObj.DLL
- PsIoDriver.tlb

These files are included in your Business Interlink SDK installation.

You must register the IoCollection.DLL and PsIntObj.DLL files in your development workstation to use the Visual Basic/COM capabilities.

To register IoCollection.DLL and PsIntObj.DLL:

1. Go to the command line. Select Start:Programs:Command Prompt.
2. Change your directory to the directory where the above files are stored.

   For example, to change to the directory containing IoCollection.DLL and PsIntObj.DLL:

   ```
   cd <PS_HOME>\bin\client\winx86
   ```

3. Execute the \winnt\system32\regsvr32.exe with each DLL (the following command unregisters before each register command as a precaution).

   ```
   C:\winnt\system32\regsvr32.exe u IoCollection.DLL
   C:\winnt\system32\regsvr32.exe IoCollection.DLL
   C:\winnt\system32\regsvr32.exe u PsIntObj.DLL
   C:\winnt\system32\regsvr32.exe PsIntObj.DLL
   ```

Setting Up Visual Basic Application Development

This section discusses how to create the C++ project that you use to write your Business Interlink plug-in under Windows NT.

To create the Visual Basic project that you use to write your Business Interlink plug-in

1. Launch Microsoft Visual Basic. Microsoft Visual Basic 6.0 is needed.
2. Select ActiveX.DLL from the new Project window. A new project appears.
3. From the main menu, select Project:References.

   The References window appears.

4. From References option menu check out the following DLLs and TLB:

   PsIoDriver.tlb. Click the Browse button to open the Add References window, navigate to the directory containing PsIoDriver.tlb, and select PsIoDriver.tlb. The directory is located at:

   ```
   <PS_HOME>\bin\client\winx86
   ```

   IoCollection.DLL and PsIntObj.DLL. You can either use the References window and check PsIntObj 1.0 Type Library and IoCollection 1.0 Type Library, or you can use the Add References window to navigate to the directory containing IoCollection.DLL and PsIntObj.DLL, and select them. The directory is located at:
   ```
   <PS_HOME>\bin\client\winx86
   ```

5. From main menu select Save Project, navigate to the directory where you want to save your project, and save it.

6. Type in “Implements PsIoDriver” in your Class (code) screen, then select PsIoDriver from the Class category pop-up list box. .
7. Select all the functions from Function category List Box (select one function at a time).

   As you select each function, the function will be enabled in the Function category list box.
8. Set the level of version compatibility of your VB project to Binary.
   a. Open your Visual Basic project.
   b. From the Project menu, select *yourproject* Properties, where *yourproject* is the name of your project (in the picture below, the project is names VbSimple).
   c. Click the Binary Compatibility radio button and click OK.
9. Implement the functions provided by PsIoDriver that you will need for your project.

   See Chapter 11, “Writing the Version Methods for a Business Interlink Runtime Plug-In,” page 135 and Chapter 12, “Writing the Execution Method for a Runtime Plug-In,” page 137.

   If there is no Business Interlink XML design-time plug-in for your project, you will need to implement some or all of the dynamic methods.

   Your runtime plug-in should use the same name as the name of the XML design-time plug-in. For example, if runtime is fcarrier.dll, the design-time should be fcarrier.xml.

Setting Up the Development Environment in Java

This section discusses how to:

- Set up your development environment for writing a Business Interlink runtime plug-in in Java under Windows NT.
- Set up your development environment for writing a Business Interlink runtime plug-in in Java under UNIX/Linux.

Setting Up the Development Environment in Java Under Windows NT

This section discusses how to create the Java project that you use to write your Business Interlink plug-in under Windows NT.

Note. For Windows NT, the Java Plug-In must be developed on, and reside on, the same system where the application server is installed.

To create the Java project that you use to write your Business Interlink plug-in under Windows NT

1. If you have not done so, install Java Development Kit version 1.4.1 and Java Runtime Environment 1.3.

   Java Development Kit version 1.4.1 is located at:

   `http://java.sun.com/products/jdk/`

   Java Runtime Environment is located at:
   `http://java.sun.com/products/jdk/jre/index.html`

2. Append the psinterlinks.jar file to your CLASSPATH variable (if you do not have one, create it).
   a. In Windows NT, right-click on the icon for your computer.
   b. Select Properties.
   c. In the System Properties panel, Environment tab, append the following text in the Value edit box.

      ```
      C:\<PS_HOME>\class\psinterlinks.jar
      ```

3. Set the PATH variable.

   If you are using a 2-tier system (you are not using an application server), then you must set the following lines in the PATH variable:

   ```
   <PS_HOME>\jre\bin
   <PS_HOME>\jre\bin\client
   ```

   If you are using a 3-tier system (you are using an application server), you must enter the following line into the PATH variable in the psappsrv.env file (located in the directory \appserv\<name of application server domain>):

   ```
   <PS_HOME>\jre\bin\client
   ```

   <location of jre> bin

4. Copy the file named psjavaproxy.dll into the InterfaceDrivers directory, which is where you will place your Java runtime plug-in, then rename the copied file to match the name of your Java runtime plug-in.

   psjavaproxy.dll is located in the InterfaceDrivers directory:

   ```
   <PS_HOME>\bin\client\winx86\InterfaceDrivers\psjavaproxy.dll
   ```

5. You can use a java file provided in the following directory to use as a template for your Java runtime plug-in.

   ```
   <PS_HOME>\sdk\PSINTERLINKS\Src\Java\Samples
   ```

6. You must append the location of your Java runtime plug-in to your CLASSPATH variable.

   If the runtime plug-in is a .class file, you need only add the directory location:

   ```
   <PS_HOME>\class\
   ```

   If the runtime plug-in is a .jar files, you must include the .jar file name as well.

   ```
   <PS_HOME>\class\yourplugin.jar
   ```

In Java, you must enclose all of the Java Business Interlink methods for your runtime plug-in within a public class that implements PSBusInterlink. For example, the following code would be the public class for a Freight Carrier class names Fcarrier.

```
Public class Fcarrier extends Object implements PSBusinterlink {
   // Include all the methods you use here: GetVersion, IsVersionCompatible,
   // ExecuteTransaction.
}
```

Your runtime plug-in should use the same name as the name of the XML design-time plug-in. For example, if runtime is fcarrier.dll, the design-time should be fcarrier.xml.

Setting Up the Development Environment in Java Under UNIX/Linux

To create the Java project that you use to write your Business Interlink plug-in under UNIX/Linux

1. If you have not done so, install Java Development Kit version 1.2.2 and Java Runtime Environment 1.2.2.

   Java Development Kit version 1.2.2 is located at:

   ```
   http://java.sun.com/products/j2se
   ```

   Java Runtime Environment is located at:
   ```
   http://java.sun.com/products/j2se
   ```

2. Copy the libpsjavaproxy file into the directory where you are developing your Java plug-in and name the copied libpsjavaproxy file to be the same as that of your Java runtime plug-in.

   The libpsjavaproxy file is located at:

   ```
   <PS_HOME>\bin\client\interfacedrivers\libpsjavaproxy
   ```

   Within that directory, you could use the following command to do the copy and naming, assuming that your plug-in is a Freight Carrier plug-in, and you want to name your runtime plug-in Fcarrier.
   ```
   cp libpsjavaproxy.so libFcarrier.so
   ```

   The libpsjavaproxy file is named:

   - on Solaris, libpsjavaproxy.so
   - on AIX, libpsjavaproxy.a
   - on HP UNIX/Linux, libpsjavaproxy.sl

3. Write your Java runtime plug-in.

   The <PS_Home>/class directory is the recommended location where you place your Java runtime plug-in. If you are writing a Freight Carrier plug-in, then your plug-in could be named Fcarrier.class.

   You can use java files provided in the following directory to use as a template for your Java runtime plug-in.

   ```
   <PS_HOME>\sdk\psinterlinks\Src\Java\Samples
   ```

   See Chapter 11, “Writing the Version Methods for a Business Interlink Runtime Plug-In,” page 135 and Chapter 11, “Writing the Version Methods for a Business Interlink Runtime Plug-In,” page 135.

   Your runtime plug-in should use the same name as the name of the XML design-time plug-in. For example, if runtime is fcarrier.dll, the design-time should be fcarrier.xml.

   In Java, you must enclose all of the Java Business Interlink methods for your runtime plug-in within a public class that implements PSBusInterlink. For example, the following code would be the public class for a Freight Carrier class names Fcarrier.
   ```
   Public class Fcarrier extends Object implements PSBusinterlink {
      // Include all the methods you use here: GetVersion, IsVersionCompatible,
      // ExecuteTransaction.
   }
   ```

See Chapter 8, “Writing an XML Design-time Plug-in,” page 101.

Using the Business Interlink SDK Directory Structure

This section discusses how to:

- Use the Business Interlink directory structures for UNIX/Linux.

- Use the Business Interlink directory structures for Windows NT.

Using the Business Interlink SDK Directory Structure: UNIX/Linux

You might need to untar the SDK directory structure. To do so, run the following commands:

```
cd $PS_HOME
./ untarsdk.sh
```

The Business Interlink directory structure on your UNIX/Linux installation consists of several directories:

- $PS_HOME, which is the location of the PeopleTools installation. This directory contains a file named psconfig.sh, and has a path leading to all the other directories listed here: $PS_HOME/sdk/*UNIXinstall*/, where *UNIXinstall* is the directory containing the UNIX SDK for your system.
- bin, which contains the file bitest, which is the executable for the UNIX/Linux Business Interlink tester.
- src/C++, which contains the C++ directories inc and samples.
  - inc contains the include files used by Business Interlink runtime plug-ins written in C++.
  - samples contains the directories newplugin and simple. These are samples of C++ runtime plug-ins.
  - simple contains the files psiodrvr.cpp, psiodrvr_simple.cpp, and psiodrvr_simple.h. psiodrvr_simple.cpp is the runtime plug-in for the simple Business Interlink. It also contains a directory named unix that contains the files makefile and makeunix.mak.
  - newplugin, which contains a directory named unix that contains the files makefile and makeunix.mak.
- src/java, which contains the samples directory. The samples directory contains the directory simple, which contains a sample of a Java runtime plug-in.

```
$PS_HOME
   psconfig.sh
   bin
      bitest
      client\winx86\interfacedrivers  (for Windows NT)
      client\interfacedrivers         (for UNIX/Linux)
         libpsjavaproxy
   sdk
      src
         C++
            inc
            samples
               newplugin
                  unix
                     makefile
                     makeunix.mak
               simple
                  psiodrvr.cpp
                  psiodrvr_simple.h
                  psiodrvr_simple.cpp
                  unix
                     makefile
                     makeunix.mak
```

```
Java
   samples
      simple
         simple.class
```

Using the Business Interlink SDK Directory Structure: Windows NT

The Business Interlink directory structure on your Windows NT installation consists of several directories:

- <PS_HOME>, which is the location of the PeopleTools installation. This directory contains all the other directories listed here.
- bin\client\winx86\InterfaceDrivers, which contains the XML design-time plug-ins and the runtime plug-ins for Business Interlinks.
- Sdk\PSINTERLINKS\Src\C++, which contains:
  - inc contains the include files used by Business Interlink runtime plug-ins written in C++.
  - samples contains the samples of C++ runtime plug-ins.
  - TEMPLATES contains the template for a C++ plug-in, psiodrvr.cpp; the template for an XML design-time plug-in, template; and the template for an XML design-time plug-in using the pshttpenable runtime plug-in.
- Sdk\PSINTERLINKS\Src\Com\Samples, which contains the samples of Visual Basic runtime plug-ins.
- Sdk\PSINTERLINKS\Src\Java\Samples, which contains the samples of Java runtime plug-ins.

CHAPTER 11

Writing the Version Methods for a Business Interlink Runtime Plug-In

This chapter discusses how to:

- Write the GetVersion method for your Business Interlink plug-in class.
- Write the IsVersionCompatible method for your Business Interlink plug-in class.
- Write the InstantiateDriverInstance method for your Business Interlink plug-in class (C++ only).

Writing GetVersion for Your Business Interlink Plug-in Class

Write the GetVersion method to supply your Business Interlink plug-in with a version number. This allows you to write future versions of your Business Interlink plug-in, and to uniquely identify each version with a number.

For the Freight Carrier Business Interlink plug-in, GetVersion is coded as follows:

```
const TCHAR * CPSCustomer::GetVersion() const
{
return _T("8.00");
}
```

For Visual Basic, the method PsIoDriver_GetVer returns the version number.

```
Private Function PsIoDriver_GetVer() As String
    '
    PsIoDriver_GetVer = "1.0.0"
    '
End Function
```

Writing IsVersionCompatible for Your Business Interlink Plug-in Class

You write the IsVersionCompatible pure virtual method to have your Business Interlink plug-in tell if it is compatible with previous versions of your XML design-time plug-in. It should compare the version number in GetVersion to the version number in the XML design-time plug-in, and it should determine if those versions are compatible.

The input parameter, version, is a version number that will be supplied by the Business Interlink Framework. The framework will call the IsVersionCompatible method and test it against every version that exists for your Business Interlink plug-in, testing it for compatibility with previous versions.

For the Freight Carrier Business Interlink plug-in, you can code IsVersionCompatible as follows:

```
BOOL  IsVersionCompatible(const TCHAR* version) { return TRUE; }

// There is only one version of the PSCustomer Business Interlink Plug-in;
// return TRUE because it is compatible with itself.
```

Suppose your XML design-time plug-in has more than one version, and your runtime plug-in will be compatible with versions 8.0 and 8.1. Then you could write IsVestionCompatible as follows:

```
BOOL  IsVersionCompatible(const TCHAR* version)
{
   if (version == _T("8.00"))
      return TRUE;
   else if (version == _T("8.10"))
      return TRUE;
   else
      return FALSE;
}
```

For Visual Basic, the method is PsIoDriver_IsVersionCompatible.

```
Private Function PsIoDriver_IsVersionCompatible(ByVal bstrVerion As String) As Long
    '
    PsIoDriver_IsVersionCompatible = True
    '
End Function
```

Writing InstantiateDriverInstance for Your Business Interlink Plug-in (C++)

With C++, you need to write the method InstantiateDriverInstance. This method is called when an instance of your class needs to be instantiated.

You do not need to create a corresponding method to destroy instances of your class; destruction of the instances of your class is automatically done for you.

For the Freight Carrier Business Interlink plug-in, InstantiateDriverInstance is coded as follows:

```
extern DLLBaseDriver* InstantiateDriverInstance();
```

CHAPTER 12

Writing the Execution Method for a Runtime Plug-In

This chapter provides an overview of the execution method for a Business Interlink Runtime plug-in and discusses how to:

- Write the execution method code to take a Business Interlink object as input.
- Write the GetObjName method to get the name of the Business Interlink object.
- Write the execution method to get the configuration parameters.
- Write the GetInputDocs and the ResetCursor methods to get the input document.
- Write the GetOutputDocs and the Clear methods to get the output document.
- Write the GetOutputParams and size methods to get the output parameter information.
- Write a loop containing the GetStatus, GetDoc, GetValue, and GetNextDoc methods to get every input document and the input values.
- Write the execution method code to call the external system.
- Write the AddDoc, AddValue, and AddNextDoc methods to add the output documents and their output values.

Understanding the Execution Method

The execution method performs the bulk of the work for a runtime plug-in. Taking the Interlink object as input, it performs the following tasks:

- Extracts the inputs from the Interlink object.
- Calls the external system to perform a certain function, passes the inputs to the external system to use as its inputs.
- Receives outputs from the external system.
- Adds the outputs to the Interlink object.

This section discusses how to write the ExecuteTransaction execution method. This is the method to write when you are creating Business Interlink Transactions, which is the most common type of Business Interlink. It uses inputs and outputs, but not criteria.

The ExecutionObjectAdd method can be coded in a similar way to ExecuteTransaction. It extracts inputs from the Business Interlink object, but does not add outputs to it.

This chapter explains the tasks to perform to write the ExecuteTransaction of a Freight Carrier runtime plug-in. The tasks, as explained here, show the code needed for the Calculate Cost transaction. You can also review listings of the code for these transactions.

See Chapter 13, “Understanding Business Interlink Runtime Plug-in Code,” ExecuteTransaction in C++, page 151; Chapter 13, “Understanding Business Interlink Runtime Plug-in Code,” ExecuteTransaction in Visual Basic, page 157 and Chapter 13, “Understanding Business Interlink Runtime Plug-in Code,” ExecuteTransaction in Java, page 162.

Writing Code To Take a Business Interlink Object as Input

The execution methods that you write all receive a Business Interlink object as an input.

In C++:

```
EIOCEXECSTATUS CPSCustomer::ExecuteTransaction(InterfaceObject * IntObj,
 const int nBatchMode)
```

In Visual Basic:

```
Private Function ExecuteTransaction(ByVal pIntObj As PsIntObj,
 ByVal lBatchMode As Long) As ENUM_EIOCEXECSTATUS
```

In Java:

```
public int ExecuteTransaction(PSJInterfaceObject intobj)
```

Note. The nBatchMode and lBatchMode parameters are not currently used.

Writing GetObjName to Get the Name of Your Business Interlink Object

Call the InterfaceObject method GetObjName to get the name of your Business Interlink object. For the ExecuteTransaction method, this retrieves the name of the transaction. This is the name set in the XML design-time plug-in.

In the Freight Carrier example, the following code tests for the transaction named “Calculate Cost(Domestic)”, or in the cast of Java, “Shipping Time”.

In C++:

```
if(IntObj->GetObjName() == _T("Calculate Cost"))
{
// Insert the transaction code here: get input parameters, call
// the external system, insert output values.
}
```

In Visual Basic:

```
If pIntObj.ObjName = "Calculate Cost" Then
```

In Java:

```
if (intobj.GetObjName().equals("Shipping Time"))
```

Writing Code To Get The Configuration Parameters

In the case of the Freight Carrier plug-in, there are no configuration parameters that are used in ExecuteTransaction.

See Also

Chapter 16, “Using The Business Interlink Methods,” GetConfigParams, page 177

Writing GetInputDocs, ResetCursor To Get The Input Document

Use the InterfaceObject method **GetInputDocs** to get the input document, which is a CBIDocs or PsBIDocs object. Then set the cursor to the top with the CBIDocs or PsBIDocs method **ResetCursor**. Setting the cursor to the top allows you to get the first of the input documents; or rather, the first set of input parameters.

In C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();
```

In Visual Basic:

```
Set objInputDocs = pIntObj.GetInputDocs
objInputDocs.ResetCursor
```

In Java:

```
Set objInputDocs = pIntObj.GetInputDocs
objInputDocs.ResetCursor
```

See Also

Chapter 16, “Using The Business Interlink Methods,” GetInputDocs, page 179

Chapter 16, “Using The Business Interlink Methods,” ResetCursor, page 235

Writing GetOutputDocs, Clear To Get The Output Document

Use the InterfaceObject method **GetOutputDocs** to get the output document, which is a CBIDocs or BIDocs object. Then clear the output document with the CBIDocs or BIDocs method **Clear**. Clearing the output values allows you to later add values and documents to a cleared document that has no previous values in it.

In C++:

```
CBIDocs docsOut = IntObj->GetOutputDocs(_T(""));
docsOut.Clear();
```

In Visual Basic:

```
Set objOutputDocs = pIntObj.GetOutputDocs
objOutputDocs.Clear
```

In Java:

```
PSJBIDocs docsOut = intobj.GetOutputDocs("");
docsOut.Clear();
```

See Also

Chapter 16, “Using The Business Interlink Methods,” GetOutputDocs, page 186

Chapter 16, “Using The Business Interlink Methods,” Clear, page 208

Writing GetOutputParams, size to Get Output Parameter Information

Use the InterfaceObject method **GetOutputParams** to retrieve information about the output parameters, which will be contained in the returned VARINFOLIST or PsEnumVarInfo object. In C++, use the VARINFOLIST method **size** to get the number of output parameters to retrieve.

The following code will get the information for the output parameters Rate and return_status by setting indexes pointing to the parameter information.

Note. Service_Rate is a document that is the output parameter for this transaction, so the names of the output parameters in the VARINFOLIST object are prefixed with “Service_Rate.”.

Note. The output parameters return_status_msg and return_status are not created at design time, unlike the other output parameters. The return_status_msg and return_status output parameters are automatically created for every set of output parameters for a Business Interlink.

In C++:

Note. The C++ example for Calculate Cost uses one output parameter called Service_Rate, type object, containing the strings Service_Type, Guaranteed_By, and Rate.

```
int iReturnStatus = -1;
int iRate = -1;

int iServiceType = -1;
int iRate = -1;
int iGuaranteed = -1;

const VARINFOLIST& varListOutput = IntObj->GetOutputParams();
for(int i = 0; i < varListOutput.size(); i++)
{
   if(varListOutput[i]->m_strName == _T("return_status_msg"))
      iReturnMessage = i;
   else if(varListOutput[i]->m_strName == _T("return_status"))
      iReturnStatus = i;
   else if(varListOutput[i]->m_strName == _T("Service_Rate.Service_Type"))
      iServiceType = i;
```

```
      else if(varListOutput[i]->m_strName == _T("Service_Rate.Guaranteed_By"))
         iGuaranteed = i;
      else if(varListOutput[i]->m_strName == _T("Service_Rate.Rate"))
         iRate = i;
   }
```

m_strName in objVarInfoName is part of the VARINFOLIST parameter list structure.

In Visual Basic:

Note. The Visual Basic example for Calculate Cost uses one output parameter called rate, type string.

```
Dim lResult As Long
Dim lIndex As Long
Dim objOutputParams As PsEnumVarInfo
Dim objVarInfo As VarInfo


Set objOutputParams = pIntObj.GetOutputParams
For Each objVarInfo In objConfigParams
   If objVarInfo.Name = "Rate" Then
      lResult = lIndex
   ElseIf objVarInfo.Name = "return_status_msg" Then
      lReturnMessage = lIndex
   ElseIf objVarInfo.Name = "return_status" Then
      lReturnStatus = lIndex
   End If
   lIndex = lIndex + 1
Next
```

Name in objVarInfoName is part of the VarInfo parameter list structure.

In Visual Basic:

```
int nCountOut = 0;
int noError = 0;
int iReturnMessage = -1
int iReturnStatus = -1
int iTime = -1
nCountOut = intobj.GetOutputParamCount();
for (int i=0; i<nCountOut; i++) {
   try
   {
      PSJVarInfo varinfo = intobj.GetOutputParam(i);
      if(varinfo.strName().equals("return_status_msg"))
         iReturnMessage = i;
      else if(varinfo.strName().equals("return_status"))
         iReturnStatus = i;
      else if(varinfo.strName().equals("Time")) iTime = i;
   }
   catch ( NoObjectReferenceException e)
   {
```

```
      return PSReturnStatus.FAILED;
   }
} //end for (int i=0; i<nCountOut; i++)
```

See Also

Chapter 16, “Using The Business Interlink Methods,” Using Parameter Lists, page 245

Chapter 16, “Using The Business Interlink Methods,” GetOutputParams, GetOutputParam, GetOutputParamCount, page 187

Chapter 16, “Using The Business Interlink Methods,” size, page 242

Writing a Loop To Get Every Input Document and the Input Values: GetStatus, GetDoc, GetValue, GetNextDoc

A Business Interlink object can have many input documents, or sets of input parameters, whose values you extract in the **ExecuteTransaction** method.

In this example, edited to show the **GetValue** methods better, use the CBIDocs or PsBIDocs method **GetStatus** method to loop through the input documents. Within the loop, use the CBIDocs or PsBIDocs method **GetDoc** to get the input parameters. Do this because the input parameters in this example are documents instead of basic types (such as integer, string, or float), so you must get the documents for these input parameters before you can get the parameter values. Then use the CBIDocs or PsBIDocs method **GetValue** to get the input values. At the end of the loop, use the CBIDocs or PsBIDocs method **GetNextDoc** to go to the next input document.

Note. You could also use the **GetCount** method to loop though each set of input parameters.

```
while(docsIn.GetStatus() == eNoError)  // loop at root level
{

   CBIDocs docFrom = docsIn.GetDoc(_T("From"));
   CBIDocs docTo = docsIn.GetDoc(_T("To"));
   CBIDocs docPackageInfo = docsIn.GetDoc(_T("Package_Info"));

   const TCHAR *pStr = docFrom.GetValue(_T("Country"));
   pStr = docFrom.GetValue(_T("Postal_Code"));

   pStr = docTo.GetValue(_T("Country"));
   pStr = docTo.GetValue(_T("Postal_Code"));
   pStr = docTo.GetValue(_T("City"));
   pStr = docTo.GetValue(_T("Address_Type"));

   pStr = docPackageInfo.GetValue(_T("Drop_off_Pickup"));
   pStr = docPackageInfo.GetValue(_T("Packaging"));
   pStr = docPackageInfo.GetValue(_T("Weight"));
   pStr = docPackageInfo.GetValue(_T("Length"));
   pStr = docPackageInfo.GetValue(_T("Width"));
   pStr = docPackageInfo.GetValue(_T("Height"));
```

```
/* Call the external system using these input values, insert the output?
 values into the output document (code not shown here) */

   docsIn.GetNextDoc();
} //end while
```

In Visual Basic:

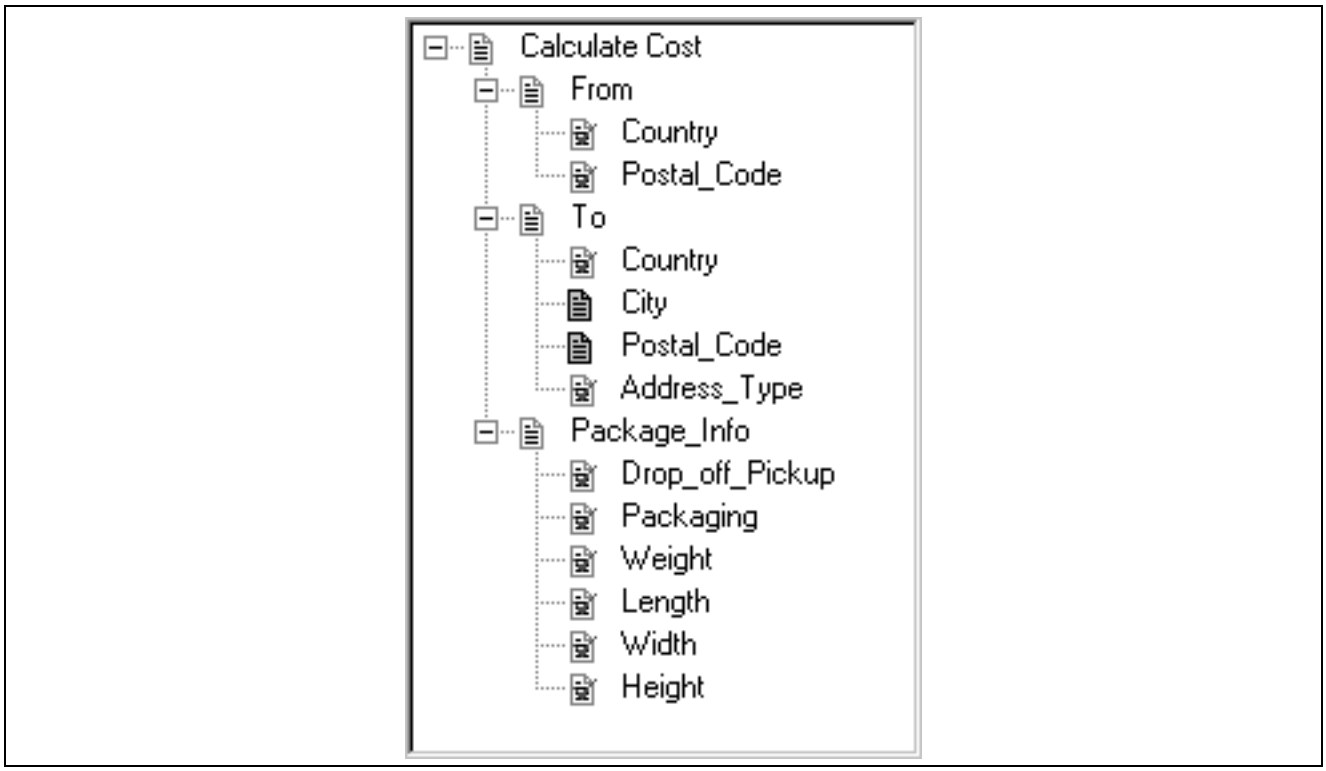
```
Dim eNoError As Integer
Dim objInputDocs As PsBIDocs
Dim objDocFrom As PsBIDocs
Dim objDocTo As PsBIDocs
Dim objDocPackInfo As PsBIDocs


While objInputDocs.GetStatus = eNoError

   Set objDocFrom = objInputDocs.GetDoc("From")
   Set objDocTo = objInputDocs.GetDoc("To")
   Set objDocPackInfo = objInputDocs.GetDoc("Package_Info")

   lOrigCountry = objDocFrom.GetValue("OriginCountry")
   lOrigPostalCode = objDocFrom.GetValue("OriginPostal_Code")

   lDestCountry = objDocTo.GetValue("DestCountry")
   lDestPostalCode = objDocTo.GetValue("DestPostal_Code")

   lWeight = objDocPackInfo.GetValue("Weight")
   lLength = objDocPackInfo.GetValue("Length")
   lWidth = objDocPackInfo.GetValue("Width")
   lHeight = objDocPackInfo.GetValue("Height")
   lPackaging = objDocPackInfo.GetValue("Packaging")
   lDrop_Off_Pickup = objDocPackInfo.GetValue("Drop_Off_Pickup")

   ' Call the external system using these input values, insert the
   ' output values into the output document (code not shown here)

   objInputDocs.GetNextDoc
Wend
```

In Java:

```
while (docsIn.getStatus() == noError) {
   docsIn.ResetCursor();
   PSJBIDocs from = docsIn.GetDoc("From");
   PSJBIDocs to = docsIn.GetDoc("To");

   String fromOriginCountry = from.GetValue("OriginCountry");
   String fromOriginPostal_Code = from.GetValue("OriginPostal_Code");
   String fromShip_Date = from.GetValue("Ship_Date");
   String toDestCountry = to.GetValue("DestCountry");
   String toDestPostal_Code = to.GetValue("DestPostal_Code");
```

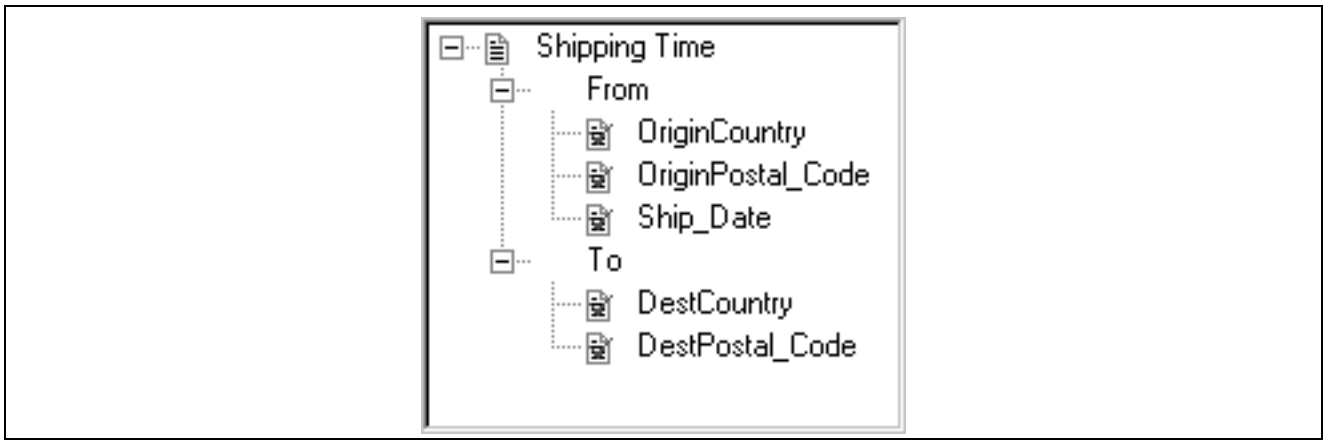
```
        docsIn.getNextDoc();

        // Call the external system using these input values, insert the
        // output values into the output document (code not shown here)
    }
    docsIn.destroy();
```

The figures below shows the input documents for this example. For Calculate Cost, the input document contains four input parameters: From, To, Package_Info, and Account_Info. For Shipping Time, the input document contains two input parameters: From and To. Since the input parameters are documents, you must get the documents if you want to get their values. The following figure shows the input document that **GetDoc** gets in this example.



Example Input Document: Calculate Cost

The following figure shows the output document that **GetDoc** gets in this example.



Example Input Document: Shipping Time

See Also

Chapter 16, “Using The Business Interlink Methods,” GetCount, page 211

Chapter 16, “Using The Business Interlink Methods,” GetDoc, page 215

Chapter 16, “Using The Business Interlink Methods,” GetNextDoc, page 218

Chapter 16, “Using The Business Interlink Methods,” GetStatus, page 227

Chapter 16, “Using The Business Interlink Methods,” GetValue, page 229

Writing Code To Call The External System

Within the loop to get the input values, call the external system, passing it the input parameter values. The code you write here depends upon your system. This will provide the values that you will add to the output documents.

Writing AddDoc, AddValue, AddNextDoc To Add The Output Documents and Their Output Values

Loop to add each output document (set of output parameters) and add the output values to the output document.

Note. In this example, for each set of input parameters, there is a corresponding set of output parameters. Therefore, the loop to add the output values is the loop to get the input values.

Use the method **AddNextDoc** to add an output document, testing with the method **Empty** to make sure that you are adding to an existing output document structure. Then use the method **AddDoc** to add the output parameters. You do this because some of the output parameters in this example are documents instead of basic types (such as integer, string, or float), so you must add the documents for these output parameters before you can add the parameter values. Then use the method **AddValue** to add the output values to the output parameters.

The output parameters return_status and return_status_msg are used in every Business Interlink object. return_status is any status number that you would like to have the Business Interlink object return; return_status_Msg is any status message that you would like to have the Business Interlink object return. They are not documents, so you do not use **AddDoc** to add them before you add their values with **AddValue**.

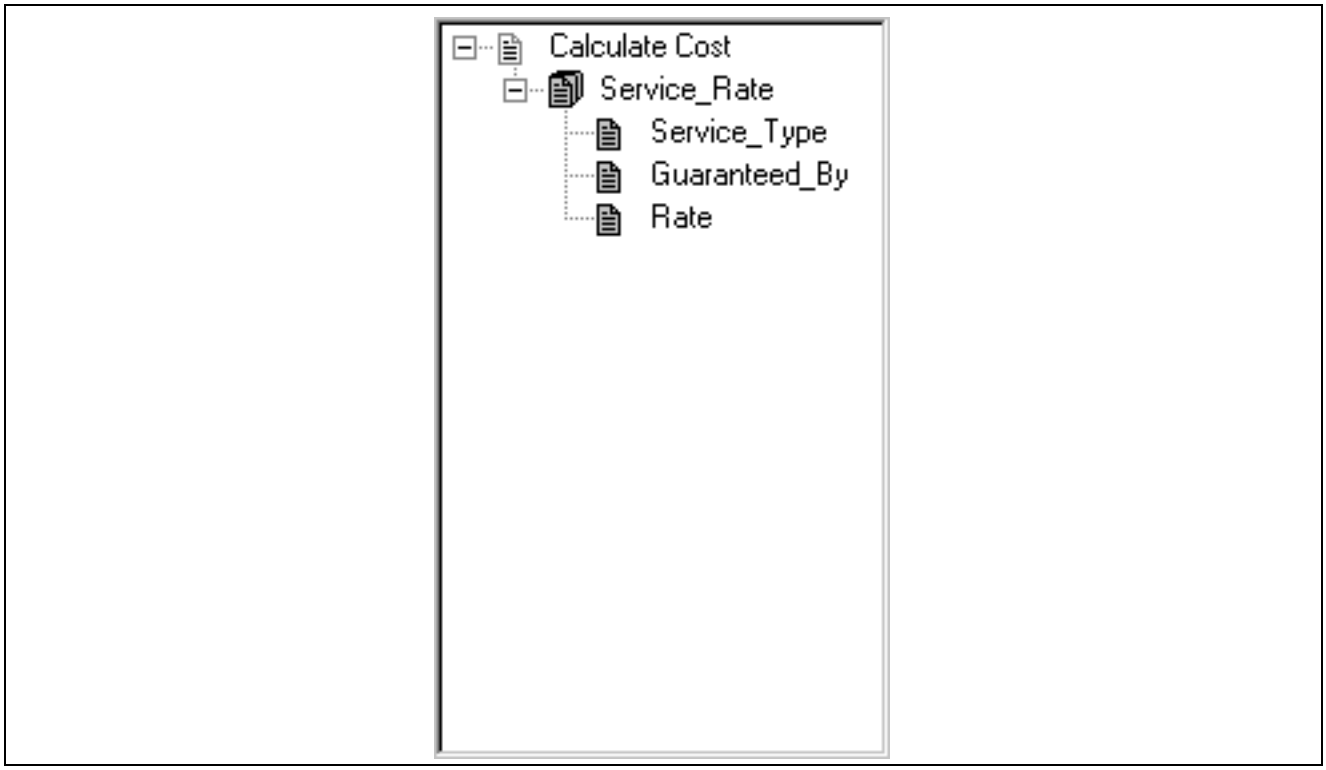
The following code adds values for Rate, which is part of the Service_Rate output parameter/document, and the output parameter return_status. This code is contained within the loop to get each set of input parameters.

Note. The **GetOutputDocs** method provided the pointer to the first output document.

```
// Loop to get a set of input parameters and call the external
// system (code not shown).

if(!docsOut.Empty()) docsOut.AddNextDoc();
   IOSTRINGLIST listServiceType, listGuaranteed, listRates;
   if(PSCustomerinfo.GetCost(listServiceType, listGuaranteed, listRates))
   {
```

```
        CBIDocs docServer = docsOut.AddDoc(_T("Service_Rate"));
           for(int i = 0; i < listServiceType.size(); i++)
           {
              if(iServiceType != -1)
                 docServer.AddValue(_T("Service_Type"),
                    listServiceType[i].c_str());
              if(iGuaranteed != -1)
                 docServer.AddValue(_T("Guaranteed_By"),
                    listGuaranteed[i].c_str());
              if(iRate)
                 docServer.AddValue(_T("Rate"), listRates[i].c_str());

              if(i < listServiceType.size() - 1)
                 docServer.AddNextDoc();
           }
        }
     }
     docsOut.AddValue(_T("return_status"), _T("0"));
     docsOut.AddValue(_T("return_status_msg"), _T("Ok"));

  // End of loop (code not shown)
```

The figure below shows the output document for the C++ example. It contains one output parameter: Service_Rate. Since the output parameters is a document, it must be added if you want to add its values. (Actually, since it is a document list, you need to add it for every document in the document list.) The figure below shows the Service_Rate document that **AddDoc** adds in the C++ example.



Example CBIDocs Output Document for C++

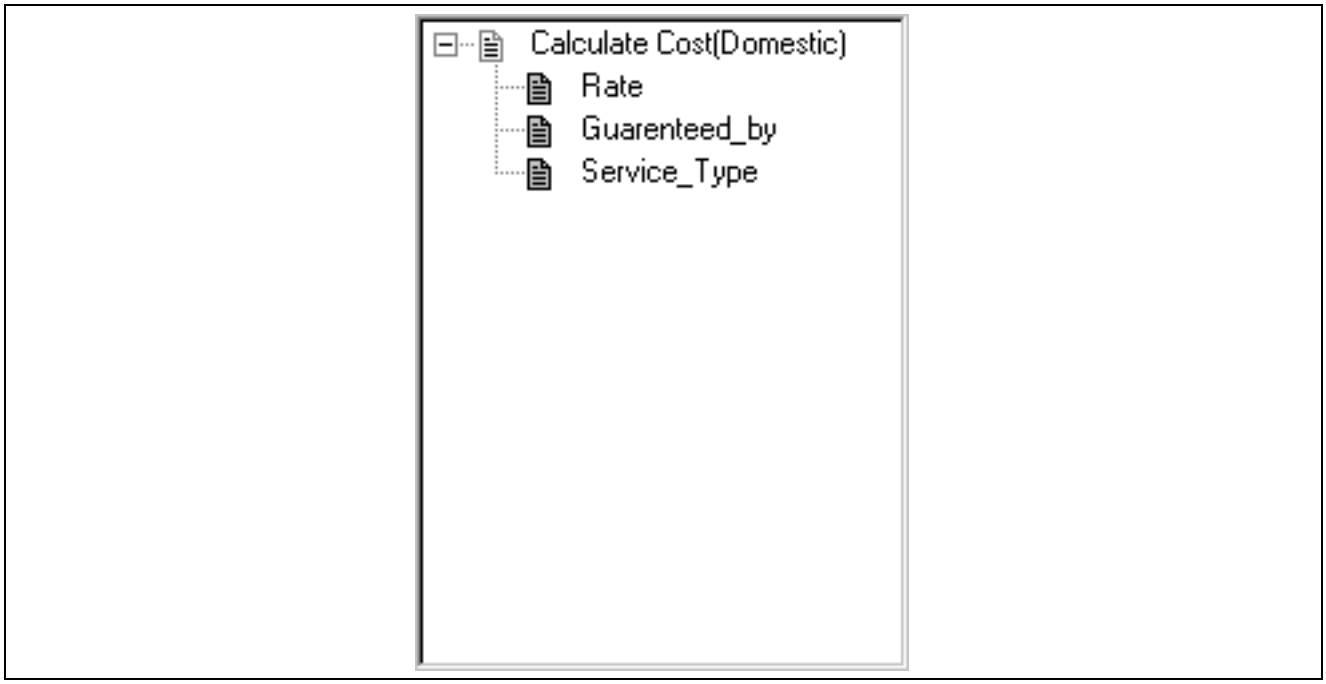
In Visual Basic:

Note. In the Visual Basic example, the output parameters are not contained in a document; they are of simple type.

```
' Set up a loop to go through all 8 service types
Dim CNT As Integer
Dim lResult As Long
Dim lResult2 As Long
Dim lService_Type As String
Dim objOutputDocs As PsBIDocs


CNT = 1
While CNT < 8

   Select Case CNT
      Case 1
         lService_Type = "Next Day Air Early A.M."
      Case 2
         lService_Type = "Next Day Air"
      ' rest of cases not shown
      End Select

   ' Set values for lRate, lGuarantee (code not shown)

   ' Test the CBIDocs output document to make sure it is not
   ' empty, and then add a document to it so you can add
   ' values to a set of output parameters.
   If objOutputDocs.Empty <> 1 Then
      lResult = objOutputDocs.AddNextDoc
   End If

   If lResult <> -1 Then
      lResult = objOutputDocs.AddValue("Rate", lRate)
      lResult = objOutputDocs.AddValue("Guarenteed_by",
                lGuarantee)
      lResult = objOutputDocs.AddValue("Service_Type",
                lService_Type)
   End If

   If lResult <> -1 Then
      lResult2 = objOutputDocs.AddValue("return_status_msg",
                 "Success")
   End If

   If lResult <> -1 Then
      lResult2 = objOutputDocs.AddValue("return_status", "0")
   End If

   CNT = CNT + 1
```

```
Wend
objInputDocs.GetNextDoc
```

The figure below shows the output document for the Visual Basic example. It contains three output parameters: Rate, Guaranteed_by, and Service_Type.

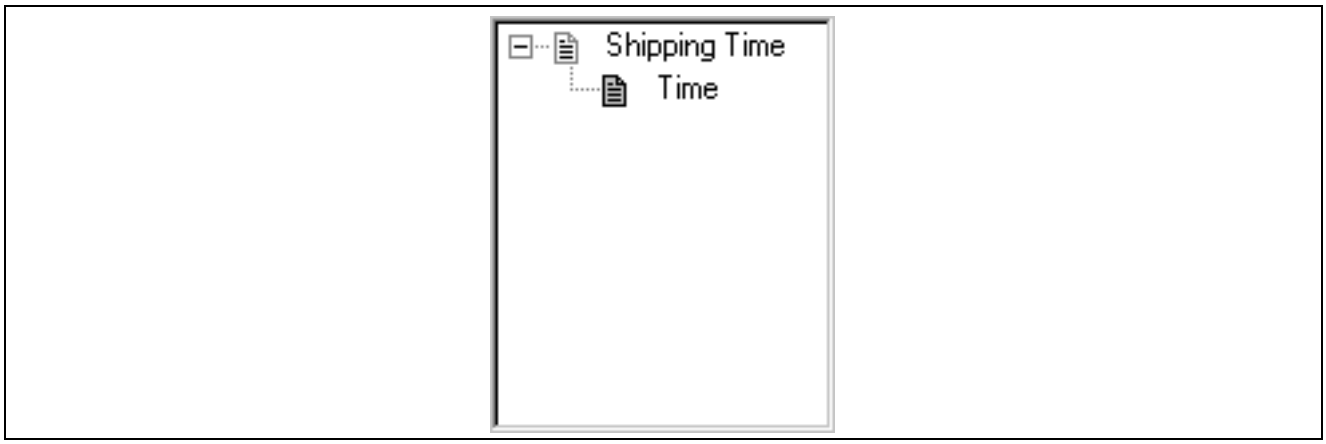


Example PsBIDocs Output Document for Visual Basic

In Visual Basic:

Note. In the Java example, the output parameters are not contained in a document; they are of simple type.

```
docsOut.AddValue("Time",sTime);
docsOut.AddValue("return_status_msg", return_status_msg);
docsOut.AddValue("return_status", "0" );
```



Example PSJBIDocs Output Document for Java

See Also

Chapter 16, “Using The Business Interlink Methods,” Empty, page 210

Chapter 16, “Using The Business Interlink Methods,” AddDoc, page 195

Chapter 16, “Using The Business Interlink Methods,” AddValue, AddValueInt, AddValueFloat, AddValueDouble, page 203

Chapter 16, “Using The Business Interlink Methods,” AddNextDoc, page 199

CHAPTER 13

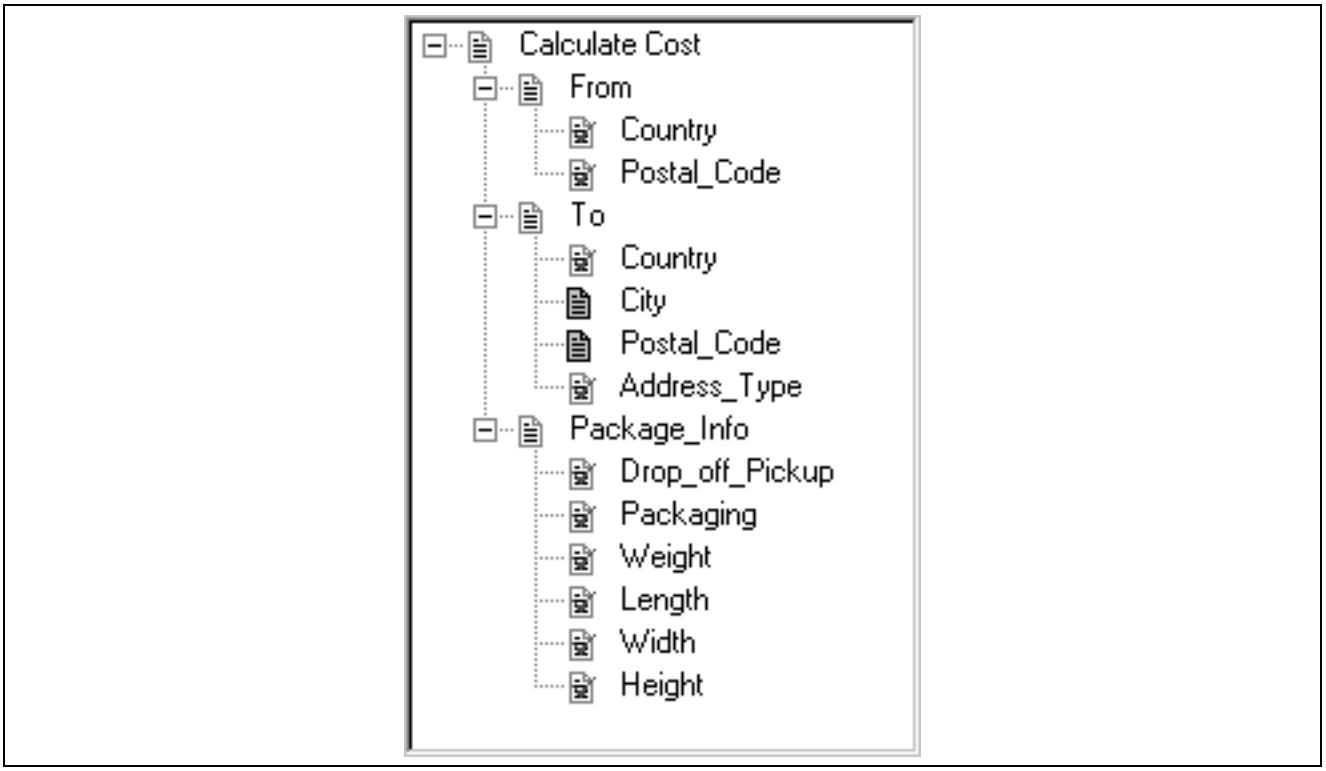
Understanding Business Interlink Runtime Plug-in Code

This chapter provides overviews of:

- An execute transaction in C++.
- An execute transaction in Visual Basic.
- An execute transaction in Java.

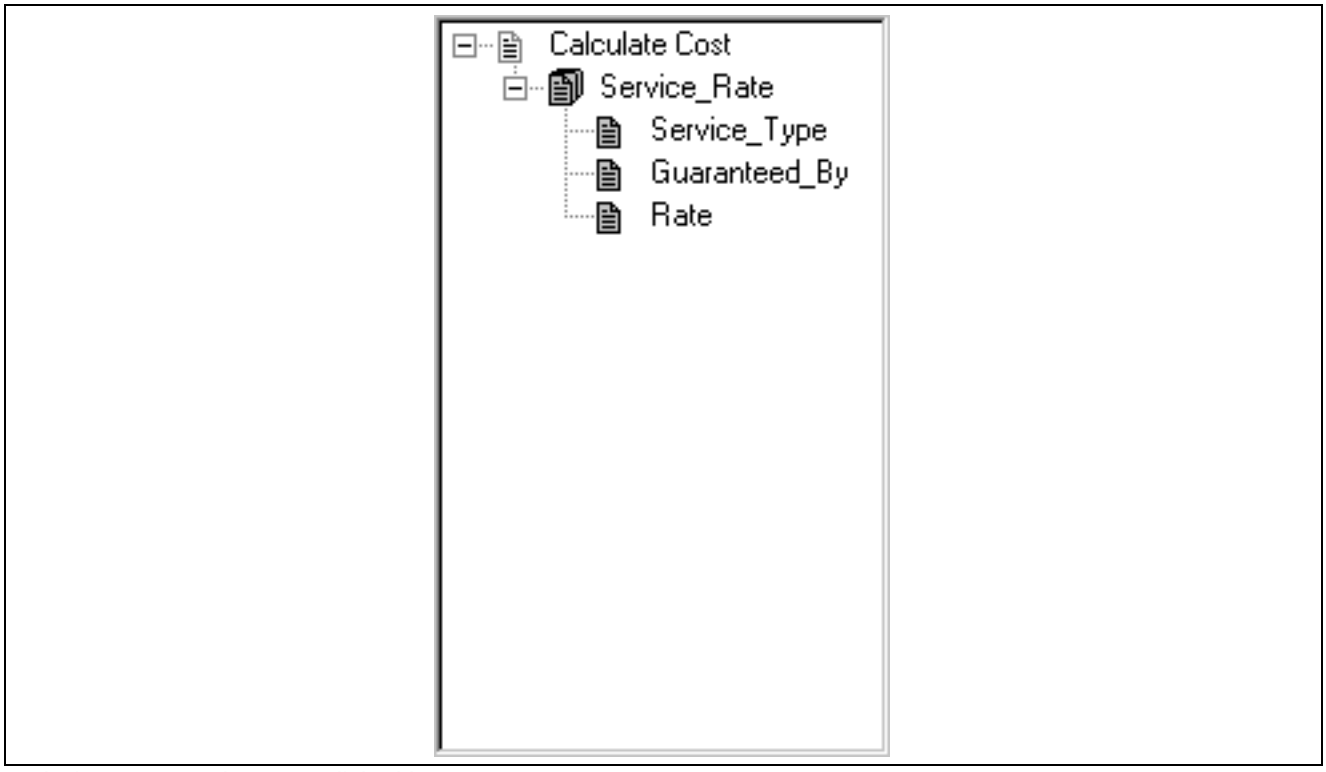
ExecuteTransaction in C++

For reference, here is the structure of the input document for the Calculate Cost Business Interlink object.



Calculate Cost Business Interlink object inputs

Here is the structure of the output document for the Calculate Cost Business Interlink object.



Calculate Cost Business Interlink object outputs

The following code shows the ExecuteTransaction for a PSCustomer plug-in. It determines which transaction is being passed in with this Business Interlink object, and processes the inputs and outputs for that transaction. The transaction is the Calculate Cost transaction.

```
EIOCEXECSTATUS CPSCustomer::ExecuteTransaction(InterfaceObject * IntObj,
 const int nBatchMode)
{

// Create the CBIDocs output document and get a reference to it, and
// then clear the output document.
   CBIDocs docsOut = IntObj->GetOutputDocs(_T(""));
   docsOut.Clear();

// Create the CBIDocs input document and get a reference to it.
   CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
   docsIn.ResetCursor();
   CTime t;
   int iReturnMessage = -1;
   int iReturnStatus = -1;

   // Get the transaction name. You need the name in order to know
   //  which inputs and outputs you will use, and how you want to
   // process them.
   if(IntObj->GetObjName() == _T("Calculate Cost"))
   {
      int iServiceType = -1;
      int iRate = -1;
```

```
      int iGuaranteed = -1;

      // Get the output parameter names.
      const VARINFOLIST& varListOutput = IntObj->GetOutputParams();
      for(int i = 0; i < varListOutput.size(); i++)
      {
         if(varListOutput[i]->m_strName == _T("return_status_msg"))
               iReturnMessage = i;
         else if(varListOutput[i]->m_strName == _T("return_status"))
               iReturnStatus = i;
         else if(varListOutput[i]->m_strName == _T("Service_Rate.Service_Type"))
               iServiceType = i;
         else if(varListOutput[i]->m_strName == _T("Service_Rate.Guaranteed_By"))
               iGuaranteed = i;
         else if(varListOutput[i]->m_strName == _T("Service_Rate.Rate"))
               iRate = i;
      }

// Start of loop to read each set of input documents (sets of input
// parameters) for this transaction.
      while(docsIn.GetStatus() == eNoError) // loop at root level
      {
// Get the input parameters that are documents (documents are
// structures: they contain parameters)
         CBIDocs docFrom = docsIn.GetDoc(_T("From"));
         CBIDocs docTo = docsIn.GetDoc(_T("To"));
         CBIDocs docPackageInfo = docsIn.GetDoc(_T("Package_Info"));


         CString strPostData =
               _T("accept_PSCustomer_license_agreement=yes");

// Get the value of Country, which is part of the From
// input parameter/document
         const TCHAR *pStr = docFrom.GetValue(_T("Country"));
         if(pStr)
         {
            strPostData += _T("&14_origCountry=");
            strPostData += GetValue(tableCountry, pStr);
         }

// Get the value of Postal_Code, which is part of the From
// input parameter/document
         pStr = docFrom.GetValue(_T("Postal_Code"));
         if(pStr)
         {
            strPostData += _T("&15_origPostal=");
            strPostData += pStr;
         }
```

```
// Get the value of Country, which is part of the To
// input parameter/document
         pStr = docTo.GetValue(_T("Country"));
         if(pStr)
         {
// Get the value of Postal_Code, which is part of the To
// input parameter/document
            strPostData += _T("&22_destCountry=");
            strPostData += GetValue(tableCountry, pStr);
         }
         pStr = docTo.GetValue(_T("Postal_Code"));
         if(pStr)
         {
            strPostData += _T("&19_destPostal=");
            strPostData += pStr;
         }
         /*
// Get the value of City, which is part of the To
// input parameter/document
         pStr = docTo.GetValue(_T("City"));
         if(pStr)
         {
            strPostData += _T("&20_origCity=");
            strPostData += pStr;
         }
         */
         strPostData += _T("&20_origCity=");
         strPostData += pStr;
// Get the value of Address_Type, which is part of the To
// input parameter/document
         pStr = docTo.GetValue(_T("Address_Type"));
         if(pStr)
         {
            strPostData += _T("&49_residential=");
            strPostData +=  GetValue(tableAddressType, pStr);
         }

// Get the value of Drop_off_Pickup, which is part of the
// Package_Info input parameter/document
         pStr = docPackageInfo.GetValue(_T("Drop_off_Pickup"));
         if(pStr)
         {
            strPostData += _T("&47_rate_chart=");
            strPostData +=  _T("Regular Daily Pickup"); // pStr;
         }

// Get the value of Packaging, which is part of the
// Package_Info input parameter/document
         pStr = docPackageInfo.GetValue(_T("Packaging"));
         if(pStr)
```

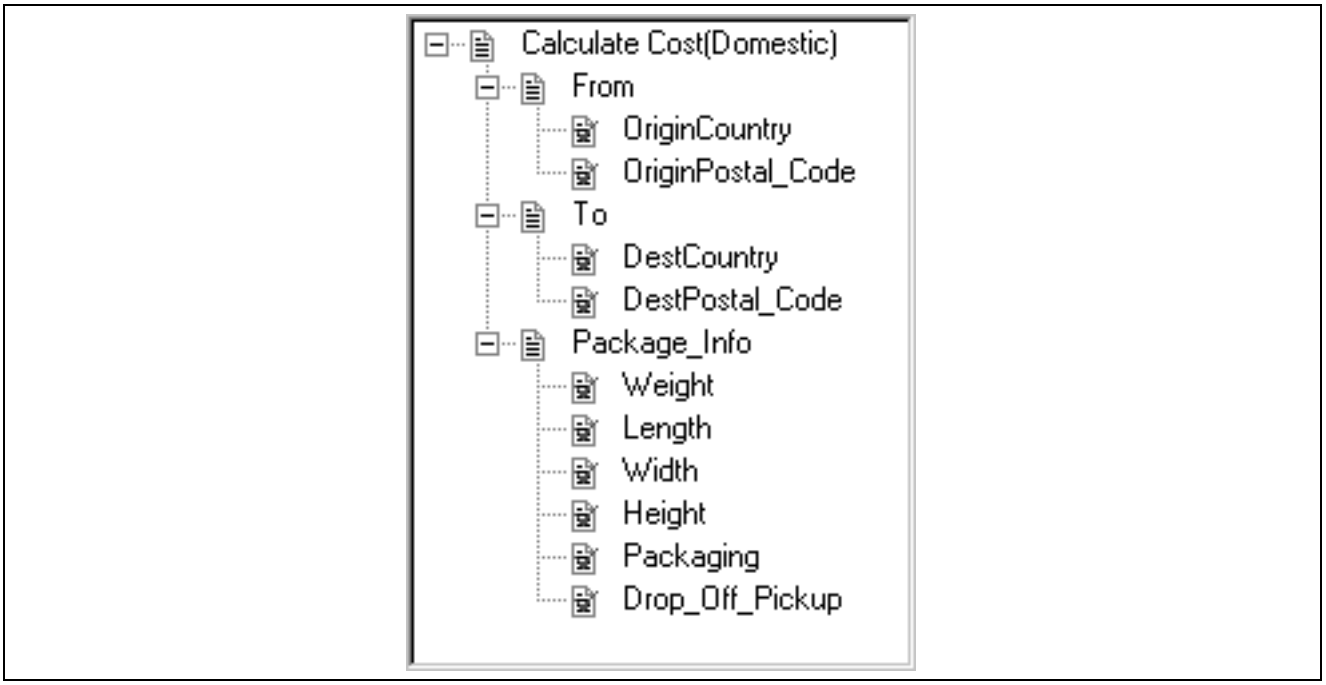
```
         {
            strPostData += _T("&48_container=");
            strPostData +=  _T("00");
               // GetValue(tableContainer, pStr);
         }

// Get the value of Weight, which is part of the
// Package_Info input parameter/document
         pStr = docPackageInfo.GetValue(_T("Weight"));
         if(pStr)
         {
            strPostData += _T("&23_weight=");
            strPostData +=  pStr;
            strPostData += _T("&weight_std=lbs.");
         }

// Get the value of Length, which is part of the
// Package_Info input parameter/document
         pStr = docPackageInfo.GetValue(_T("Length"));
         if(pStr)
         {
           strPostData += _T("&25_length=");
            strPostData +=  pStr;
         }

// Get the value of Width, which is part of the
// Package_Info input parameter/document
         pStr = docPackageInfo.GetValue(_T("Width"));
         if(pStr)
         {
            strPostData += _T("&26_width=");
            strPostData +=  pStr;
         }

// Get the value of Height, which is part of the
// Package_Info input parameter/document
         pStr = docPackageInfo.GetValue(_T("Height"));
         if(pStr)
         {
            strPostData += _T("&27_height=");
            strPostData +=  pStr;
         }

         strPostData += _T("&length_std=in.");
         strPostData.Replace(_T(" "), _T("+"));

         CInet iNet;
         PSIOString strFileName = _T("c:\\temp\\abc.html");
            //GenerateTempFileName();
```

```
// Execute the transaction by calling the external system
         if(iNet.PostRequest(
             _T("http://www.PSCustomer.com/using/services/rave/qcost.cgi"),
             (TCHAR*)(LPCTSTR)strPostData,
             strPostData.GetLength(),
             (TCHAR *)strFileName.c_str()))
         {
            CPSCustomerInfo PSCustomerinfo(strFileName);
// Test the output document to make sure it is not empty,
// and then add a document to it so you can add values to a set of
// output parameters.
            if(!docsOut.Empty()) docsOut.AddNextDoc();

            IOSTRINGLIST listServiceType, listGuaranteed,
               listRates;
            if(PSCutomerinfo.GetCost(listServiceType, listGuaranteed,
               listRates))
            {
// Since the output parameter Service_Rate is a document, add a
// Service_Rate document to allow its values to be added.
               CBIDocs docServer =
                     docsOut.AddDoc(_T("Service_Rate"));
// Since Service_Rate is a document list, loop to add all of the
// Service Rate data that you have.
               for(int i = 0; i < listServiceType.size(); i++)
               {
// Add a value for Service_Type, which is part of the Service_Rate
// output parameter/document.
                  if(iServiceType != -1)
                     docServer.AddValue(_T("Service_Type"),
                     listServiceType[i].c_str());
// Add a value for Guaranteed_By, which is part of the Service_Rate
// output parameter/document.
                  if(iGuaranteed != -1)
                     docServer.AddValue(_T("Guaranteed_By"),
                     listGuaranteed[i].c_str());
// Add a value for Rate, which is part of the Service_Rate
// output parameter/document.
                  if(iRate)
                     docServer.AddValue(_T("Rate"),
                     listRates[i].c_str());

// Add another Service_Rate output parameter/document to the
// document list.
                  if(i < listServiceType.size() - 1)
                     docServer.AddNextDoc();
               }   // end for loop to add all of the Service Rate
                    // data that you have.
            }   // end if (PSCutomerinfo.GetCost)
        }  //end if (execute transaction)
```

```
            // Add values for the output parameters return_status_msg
            // and return_status.
            docsOut.AddValue(_T("return_status"), _T("0"));
            docsOut.AddValue(_T("return_status_msg"), _T("Ok"));
            docsIn.GetNextDoc();
         }  // end of while loop to read each set of input documents
      }  // end of if that gets the transactions name (Calculate Cost)
      return INTOBJ_SUCCEED;
   }
```

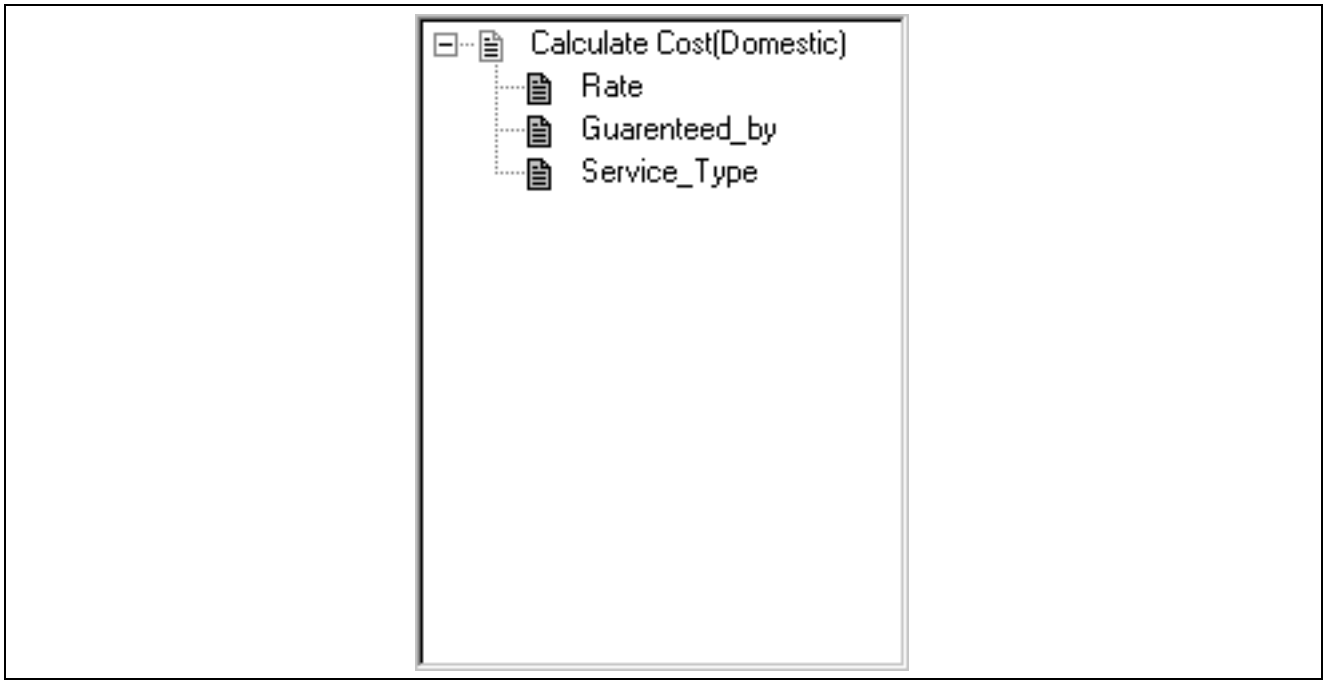
ExecuteTransaction in Visual Basic

For reference, here is the structure of the input document for the Calculate Cost (Domestic) Business Interlink object.



Calculate Cost(Domestic) Business Interlink object inputs

Here is the structure of the output document for the Calculate Cost(Domestic) Business Interlink object.



Calculate Cost(Domestic) Business Interlink object outputs

The following code shows the ExecuteTransaction for a Freight Carrier plug-in. It determines which transaction is being passed in with this Business Interlink object, and processes the inputs and outputs for that transaction. The transaction is the Calculate Cost(Domestic) transaction.

```
Private Function ExecuteTransaction(ByVal pIntObj As PsIntObj,?
 ByVal lBatchMode As Long) As ENUM_EIOCEXECSTATUS
    '
    Dim bRtnVal As Boolean
    Dim objVariantInfo As Variant
    Dim objVarInfo As VarInfo
    Dim objOutputParams As PsEnumVarInfo
    Dim lResult As Long
    Dim lResult2 As Long
    Dim lIndex As Long
    Dim lReturnMessage As Long
    Dim lReturnStatus As Long
    Dim objOutputDocs As PsBIDocs
    Dim objInputDocs As PsBIDocs
    Dim objDocFrom As PsBIDocs
    Dim objDocTo As PsBIDocs
    Dim objDocPackInfo As PsBIDocs
    Dim objDocService As PsBIDocs
    Dim objPsBstr As PsBstr
    Dim eNoError As Integer
    eNoError = 0
    bRtnVal = True
    lIndex = 0
    lResult = -1
    lReturnMessage = -1
```

```
  lReturnStatus = -1
  '

' Create the CBIDocs output document and get a reference to it,
' and then clear the output document.
  Set objOutputDocs = pIntObj.GetOutputDocs
  objOutputDocs.Clear

' Get the output parameter information.
  Set objOutputParams = pIntObj.GetOutputParams
  For Each objVarInfo In objOutputParams
      If objVarInfo.Name = "Rate" Or objVarInfo.Name = "Time" Then
          lResult = lIndex
      ElseIf objVarInfo.Name = "return_status_msg" Then
          lReturnMessage = lIndex
      ElseIf objVarInfo.Name = "return_status" Then
          lReturnStatus = lIndex
      End If
      lIndex = lIndex + 1

  Next

  Dim pValue As String
  Dim iSum As Integer
  Dim iDResult As Integer
  Dim varValue1 As Variant
  Dim varValue2 As Variant
  Dim varValue3 As Variant

  Dim lOrigCountry As String
  Dim lOrigPostalCode As String
  Dim lDestCountry As String
  Dim lDestPostalCode As String
  Dim lLength As String
  Dim lWeight As String
  Dim lWeight_Type As String
  Dim lWidth As String
  Dim lHeight As String
  Dim lPackaging As String
  Dim lDrop_Off_Pickup As String

' Get the name of the Calculate Cost transaction
  If pIntObj.ObjName = "Calculate Cost" Then
      Dim lRate As String
      Dim lGuarantee As String
      Dim lService_Type As String

    ' Create the CBIDocs input document and get a reference to it.
      Set objInputDocs = pIntObj.GetInputDocs
      objInputDocs.ResetCursor
```

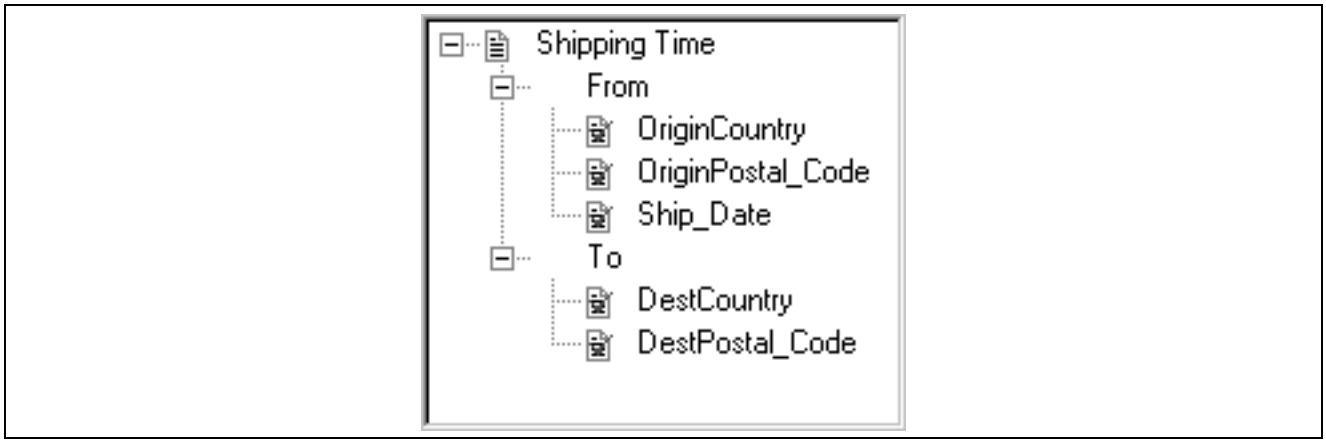
```
' Get the input parameters that are documents

 ' Start of loop to read each set of input documents (sets of
 ' input parameters) for this transaction.
  While objInputDocs.GetStatus = eNoError

     Set objDocFrom = objInputDocs.GetDoc("From")
     Set objDocTo = objInputDocs.GetDoc("To")
     Set objDocPackInfo = objInputDocs.GetDoc("Package_Info")

     lRate = ""
     lGuarantee = ""
     lService_Type = ""

   ' Get the value of OriginCountry, OriginPostal_Code, and
   ' Ship_Date, which are part of the From input
   ' parameter/document
     lOrigCountry = objDocFrom.GetValue("OriginCountry")
     lOrigPostalCode = objDocFrom.GetValue("OriginPostal_Code")

   ' Get the value of DestCountry and DestPostal_Code, which
   ' are part of the To input parameter/document
     lDestCountry = objDocTo.GetValue("DestCountry")
     lDestPostalCode = objDocTo.GetValue("DestPostal_Code")

   ' Get the value of Weight, Weight_Type, Declared_Value, and
   ' Service_Type, which are part of the Package_Info input
   ' parameter/document
     lWeight = objDocPackInfo.GetValue("Weight")
     lLength = objDocPackInfo.GetValue("Length")
     lWidth = objDocPackInfo.GetValue("Width")
     lHeight = objDocPackInfo.GetValue("Height")
     lPackaging = objDocPackInfo.GetValue("Packaging")
     lDrop_Off_Pickup = objDocPackInfo.GetValue("Drop_Off_Pickup")

     'lRate = CInt(Left(lOrigPostalCode, 1) &
     'Right(lDestPostalCode, 1))
      Dim CNT As Integer
      CNT = 1
      While CNT < 8

      Select Case CNT
      Case 1
          lService_Type = "Next Day Air Early A.M."
      Case 2
          lService_Type = "Next Day Air"
      Case 3
          lService_Type = "Next Day Air Saver"
      Case 4
```

```
        lService_Type = "2nd Day Air Early A.M."
    Case 5
        lService_Type = "2nd Day Air"
    Case 6
        lService_Type = "3rd Day Select"
    Case 7
        lService_Type = "Ground"
    End Select

    '*******************************************************

    Set objFreight = New Freight

    objFreight.OriginCountry = lOrigCountry
    objFreight.OriginPostalCode = lOrigPostalCode
    objFreight.DestCountry = lDestCountry
    objFreight.DestPostalCode = lDestPostalCode

    objFreight.ServiceType = lService_Type
    objFreight.Weight = lWeight
    objFreight.Length = lLength
    objFreight.Width = lWidth
    objFreight.Height = lHeight
    objFreight.Packaging = lPackaging
    objFreight.Drop_Off_Pickup = lDrop_Off_Pickup

    lRate = objFreight.Cost
    lGuarantee = objFreight.Guarantee_By

    '*******************************************************

' Test the CBIDocs output document to make sure it is not
' empty, and then add a document to it so you can add
' values to a set of output parameters.
    If objOutputDocs.Empty <> 1 Then
      lResult = objOutputDocs.AddNextDoc
    End If

    If lResult <> -1 Then

      ' Add values for the output parameters Rate,
      ' Guarenteed_by, and Service_Type.
      lResult = objOutputDocs.AddValue("Rate", lRate)
      lResult = objOutputDocs.AddValue("Guarenteed_by", lGuarantee)
      lResult = objOutputDocs.AddValue("Service_Type", lService_Type)

    End If
      ' Add values for the output parameters
      ' return_status_msg and return_status.
    If lResult <> -1 Then
```

```
            lResult2 = objOutputDocs.AddValue("return_status_msg", "Success")
          End If
          If lResult <> -1 Then
            lResult2 = objOutputDocs.AddValue("return_status", "0")
          End If


          CNT = CNT + 1
        Wend
          objInputDocs.GetNextDoc
        Wend



    End If

    ExecuteTransaction = ENUM_INTOBJ_SUCCEED
    '
End Function
```

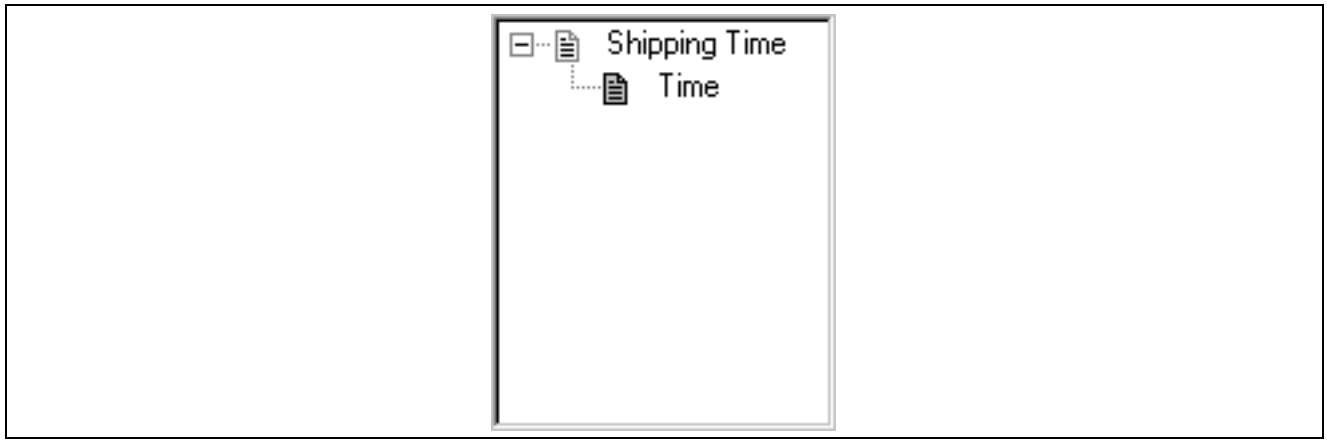
ExecuteTransaction in Java

For reference, here is the structure of the input document for the Shipping Time Business Interlink object.



Shipping Time Business Interlink object inputs

Here is the structure of the output document for the Shipping Time Business Interlink object.



Shipping Time Business Interlink object outputs

The following code shows the ExecuteTransaction for a Freight Carrier plug-in. It determines which transaction is being passed in with this Business Interlink object, and processes the inputs and outputs for that transaction. The transaction is the Shipping Time transaction.

```
public int ExecuteTransaction(PSJInterfaceObject intobj){

   String varstring = new String();

   String return_status_msg = new String("Succeded");
   int nCountOut = 0;
   int noError = 0;
   // Get the output parameters.
   nCountOut = intobj.GetOutputParamCount();
   for (int i=0; i<nCountOut; i++) {
      try
      {
         PSJVarInfo varinfo = intobj.GetOutputParam(i);
      }
      catch ( NoObjectReferenceException e)
      {
         return PSReturnStatus.FAILED;
      }
   } //end for (int i=0; i<nCountOut; i++)

   // if transaction is "Shipping Time"
   // Get the transaction name. You need the name in order to know
   //  which inputs and outputs you will use, and how you want to
   // process them.
   if (intobj.GetObjName().equals("Shipping Time"))
   {
      int ship_time;
      ship_time=0;
      // Standard initialization code.  PSJBIDocs allocate
      // memory on the C++ side so we must be sure to call the
      // destroy routine prior to leaving scope.
      try
```

```
      {
         // Create the CBIDocs input document and get a reference to it.
         PSJBIDocs docsIn = intobj.GetInputDocs("");
      }
      catch ( NoObjectReferenceException e)
      {
         return_status_msg = "Failed to get Input";
         return PSReturnStatus.FAILED;
      }
// Start of loop to read each set of input documents (sets of input
// parameters) for this transaction.
      while (docsIn.getStatus() == noError) {
         docsIn.ResetCursor();
         // Get the input parameters that are documents (documents are
         // structures: they contain parameters)
         PSJBIDocs from = docsIn.GetDoc("From");
         PSJBIDocs to = docsIn.GetDoc("To");

// Get the values of OriginCountry, OriginPostal_Code, and Ship_Date, which are
// part of the From input parameter/document
         String fromOriginCountry = from.GetValue("OriginCountry");
         String fromOriginPostal_Code = from.GetValue("OriginPostal_Code");
         String fromShip_Date = from.GetValue("Ship_Date");

// Get the values of DestCountry and DestPostal_Code, which are
// part of the To input parameter/document
         String toDestCountry = to.GetValue("DestCountry");
         String toDestPostal_Code = to.GetValue("DestPostal_Code");

// Execute the transaction by calling the external system
         ship_time = shipTime(
            toDestCountry,toDestPostal_Code,fromOriginCountry,
            fromOriginPostal_Code);

         // process output definition
         try {
            // Create the PSJBIDocs output document and get a reference to it,
            // and then clear the output document.
            PSJBIDocs docsOut = intobj.GetOutputDocs("");
            docsOut.Clear();
            String sTime = java.lang.Integer.toString(ship_time);

            // Fill in the root entry
// Add a value for Time, which is an output parameter.
            docsOut.AddValue("Time",sTime);
// Add values for the output parameters return_status_msg
// and return_status.
            docsOut.AddValue("return_status_msg", return_status_msg);
            docsOut.AddValue("return_status", "0" );
```

```
            // We explicitly call this to cleanup C++ memory behind the
            // Java Doc objects which were created during Java processing.
            // The other objects, such as the PSJInterface object are allocated
            // by the caller (framework) and is just wrappered for use here so
            // we are not responsible for freeing that memory.
            docsOut.destroy();
         }
         catch ( NoObjectReferenceException e)
         {
            return PSReturnStatus.FAILED;
         }
         docsIn.getNextDoc();
      }
      //end while that loops through input document
      docsIn.destroy();

   }  //end if transaction is "Shipping Time"

   return PSReturnStatus.OK;
}//end Execute()

public int shipTime ( String country_to, String zip_to, String country_from,?
 String zip_from)
{
   int zip_diff= java.lang.Integer.parseInt(zip_from, 10) -?
 java.lang.Integer.parseInt(zip_to, 10);
   int country_diff= country_to.compareTo( country_from);

   if (country_diff == 0){

      if (zip_diff == 0)
         return 1;
      else
         return zip_diff* 2;
      }
   else
   {
      return (zip_diff + country_diff) * 2;
   }
}
```

CHAPTER 14

Deploying a Business Interlink Runtime Plug-in

This chapter discusses how to:

- Place a Business Interlink runtime plug-in on a PeopleSoft client for testing.
- Deploy a Business Interlink on a PeopleSoft application server for others to use.
- Deploy a Business Interlink on a Windows NT web server for others to use.

Placing on PeopleSoft Application Clients for Testing

On Windows NT, if you want to test your Business Interlink runtime plug-in using the Business Interlink tester, you must place it on the PeopleSoft application client.

To place your Business Interlink runtime plug-in on a PeopleSoft application client, ensure that its DLL file (or equivalent if you are using UNIX/Linux) is located in the following location:

```
<PS_HOME>\bin\client\winx86\interfacedrivers
```

Where <PS_Home> is the directory where PeopleTools is installed.

See Also

Chapter 3, “Designing a Business Interlink Definition,” Testing Your Business Interlink Definition, page 17

Deploying on PeopleSoft Application Servers

To deploy your Business Interlink runtime plug-in on a PeopleSoft application server, ensure that its DLL file (or equivalent if you are using UNIX/Linux) is placed into the following location:

```
<PS_Home>\bin\client\winx86\interfacedrivers
```

Where <PS_Home> is the directory where PeopleTools is installed.

Deploying on Web Servers

To deploy your Business Interlink runtime plug-in on a Windows NT web server where the Business Interlink runtime environment has been installed, ensure that its DLL file is placed into the following location:

```
<WebServerRoot>\PSInterlinks\InterfaceDrivers
```

If the default location was not used when Business Interlink runtime environment was installed, contact the person who installed it to get the location.

For this deployment to work, your 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.

CHAPTER 15

Testing a Business Interlink Plug-in

This chapter provides an overview of Business Interlink testing and discusses how to:

- Use the Business Interlink tester on Windows NT.
- Use the Business Interlink tester on UNIX/Linux.
- Write an Input Doc XML file for the UNIX/Linux Business Interlink tester.

Understanding Testing of Business Interlinks

The Business Interlink Tester allows you to test your Business Interlink. If you are creating a Business Interlink runtime plug-in, you can test it using the Business Interlink Tester. If you have PeopleTools installed, you can design a Business Interlink definition and use the Business Interlink Tester within the PeopleTools Application Designer to test the Business Interlink definition. In each case, the Business Interlink Tester executes the runtime plug-in, sending it input values and then receiving output values.

Using the Business Interlink Tester: Windows NT

In order to run the Business Interlink Tester for Windows NT, you must have deployed both the runtime plug-in and the XML design-time plug-in. The instruction for running the Business Interlink tester for Windows NT are in the PeopleSoft Business Interlink Application Developer Guide.

Using the Business Interlink Tester: UNIX/Linux

Once you have set up Business Interlinks on UNIX/Linux, you can test your Business Interlink runtime plug-ins using the Business Interlink tester for UNIX/Linux. You can test runtime plug-ins that you write, and you can test the simple runtime plug-in.

To run the Business Interlink tester for UNIX/Linux:

1. Create an Input Doc XML file.
2. Save the Business Interlink XML design-time plug-in file and the Input Doc XML file into the <PS_HOME>/bin directory.
3. Within the <PS_HOME>/bin directory, run the following command to run the Business Interlink Tester for UNIX/Linux.

```
bitest  designplugin.xml inputdoc.xml [output.xml]
```

If any of the XML files are not in the same directory as the bitest executable, you must specify the path for them. If you are not currently in the same directory as the bitest executable, you must specify the path for it.

The 1st parameter, *designplugin.xml*, is the Business Interlink XML design time plug-in (or XML design-time plug-in). The file named simple.xml, which is provided and un-tared with the setting up Business Interlinks on UNIX/Linux instructions, is an XML design-time plug-in.

The 2nd parameter, *inputdoc.xml*, is an Input Doc XML file. The files named addnumber.xml, catstr.xml, and machine.xml, which are provided and un-tared with the setting up Business Interlinks on UNIX/Linux instructions, are all Input Doc XML files.

The 3rd parameter, *output.xml*, is the Output Doc XML file, where the output values from the execution of the runtime plug-in will be stored. This parameter is optional; if you do not supply it, the default file name is output.xml. If you provide a path with the file name, the Output Doc XML file will be located in that path. Otherwise, the file will be located in the bin directory (the same directory as bitest.exe).

Writing an Input Doc XML File (UNIX/Linux Business Interlink Tester)

This section provides an overview of Input Doc XML files and discusses how to:

- Write the main tag for the Input Doc XML file <Business Interlink>.
- Write the definition for the Input Doc XML file <Definition, <Header>, <InterfaceName>.
- Write the input values for the Input Doc XML file <Inputs>.

Understanding Input Doc XML Files

The Input Doc XML file contains the input values that the Business Interlink Tester needs to execute the runtime plug-in and return output values.

Here is the XML design-time plug-in code that contains a transaction named “add numbers”. This transaction adds two numbers.

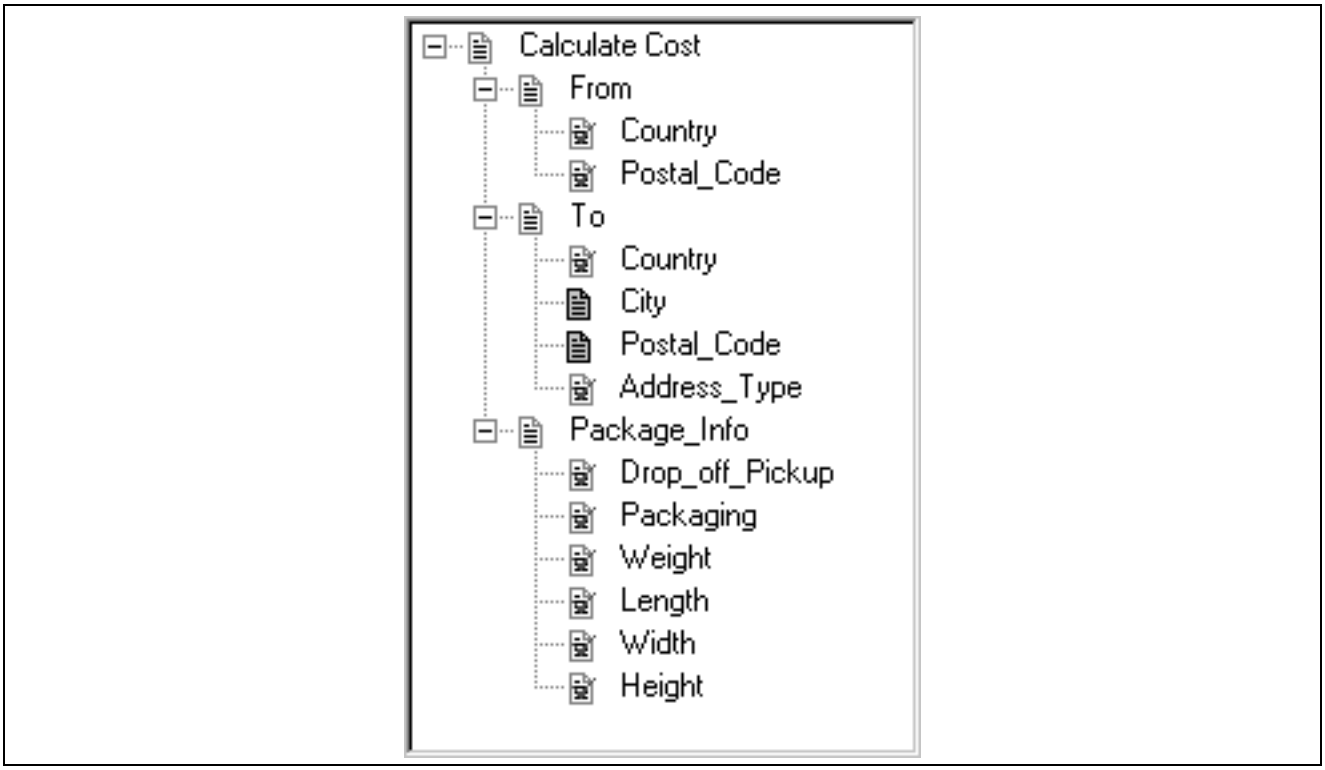
```
<trans_catalog>
   <category name="simple transactions">
      <transaction name="add numbers">
         <input_list>
            <input name="number_1" type="int" required="true"/>
            <input name="number_2" type="int" />
         </input_list>
         <output_list>
            <output name="sum" type="int"/>
         </output_list>
      </transaction>
   </category>
</trans_catalog>
```

Here is the corresponding Input Doc XML file that adds input data for the “add numbers” transaction.

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

```
<Business_Interlink>
   <Definition>
      <Header>
         <InterfaceName Value="Untitled">
         </InterfaceName>
      </Header>
  </Definition>
   <Inputs>
      <number_1>3</number_1>
      <number_2>23</number_2>
   </Inputs>
</Business_Interlink>
```

Here is an example of an input doc for the Freight Carrier plug-in. It shows how to set the Inputs tag for the following structure:



Calculate Cost Business Interlink object inputs

```
<?xml version="1.0"?>
<Business_Interlink>
   <Definition>
      <Header>
         <InterfaceName Value="Test">
         </InterfaceName>
      </Header>
   </Definition>
   <Inputs>
      <root>
         <From>
```

```
            <Country>United States</Country>
            <Postal_Code>05550</Postal_Code>
         </From>
         <To>
            <Country>United States</Country>
            <City>San Mateo</City>
            <Postal_Code>94404</Postal_Code>
         </To>
         <Package_Info>
            <Drop_off_Pickup>Regular Daily Pickup</Drop_off_Pickup>
            <Packaging>Your Packaging</Packaging>
            <Weight>1</Weight>
            <Length>4</Length>
            <Width>4</Width>
            <Height>4</Height>
          </Package_Info>
      </root>
   </Inputs>
</Business_Interlink>
```

Writing the main Tag for the Input Doc <Business_Interlink>

The main tag for the Business Interlink XML design-time plug-in is <**Business_Interlink**>.
It contains the <**Definition**> and <**Inputs**> tags.

Write the Definition for the Input Doc <Definition>, <Header>, <InterfaceName>

In the <**Definition**> tag, write the <**InterfaceName**> tag to contain the name of this Business Interlink definition.

Here is an example of how to set the <**Definition**>, <**Header**>, and <**InterfaceName**> tags.

```
<Definition>
   <Header>
      <InterfaceName Value="Untitled">
      </InterfaceName>
   </Header>
</Definition>
```

Write the Input Values for the Input Doc <Inputs>

In the <**Inputs**> tag, write a tag to set the values for each input parameter.

Here is an example of how to set the <**Inputs**> tag. It sets the values of the input parameters number_1 and number_2 to 3 and 23.

```
<Inputs>
   <number_1>3</number_1>
   <number_2>23</number_2>
</Inputs>
```

CHAPTER 16

Using The Business Interlink Methods

This chapter provides information about:

- InterfaceObject class methods, which you write for your execution method (or methods) for your Interlink class.
- Runtime plug-in class methods, which you write when you are creating a runtime plug-in. The execution methods are in this class.
- BIDocs methods, which you use to access the input and output values of a Business Interlink object when the input and output is hierarchical.
- PSIOString methods, which you use to access the information in a PSIOString, which is how the input and output parameter information (names, data types) for a Business Interlink object are stored.

InterfaceObject Class Methods

The InterfaceObject Class contains the methods that you use when you write your execution method (or methods) for your Interlink class. A Business Interlink object is passed to your execution methods, and you use the InterfaceObject class methods to retrieve information about the Business Interlink object.

You must include the header file iocls.h for this class.

The following list shows which InterfaceObject methods you use to retrieve Business Interlink object information.

- Use GetDescription() or GetDesc to retrieve a text description of the Business Interlink object
- Use GetInterfaceName() to retrieve the name of the Business Interlink definition.
- Use GetConfigParam(strParamName) to retrieve the value of one configuration parameter.
- Use GetConfigParams() to retrieve the list of configuration parameters for this Business Interlink object.
- Use GetInputParams() to retrieve the input parameter information for this Business Interlink object.
- Use GetInputTable() to retrieve the input table containing the input values for this Business Interlink object.
- Use GetInterfaceType() to retrieve the type of this Business Interlink object.
- Use GetObjName() to retrieve the name of this Business Interlink object.
- Use GetOutputParams() to retrieve the output parameter information for this Business Interlink object.
- Use GetOutputTable() to retrieve the output table into which you will insert output values for this Business Interlink object.

ConfigParam

Description

Refer to GetConfigParam for this method.

Description

Description

Refer to GetDescription for this method.

GetDescription, Description

Syntax

C++:
```
PSIOString& GetDescription()
```

Visual Basic:
```
string Description
```

Java:
```
String GetDescription()
```

Description

The C++ **GetDescription** method and the Visual Basic property **Description** get the description of this Business Interlink plug-in. This description is set when you write the XML design-time plug-in or write the **GetDesc** method.

The classes are:

- C++: InterfaceObject
- Visual Basic: IpsIntObj
- Java: PSJInterfaceObject

Parameters

None.

Returns

A string containing the description.

Value	Meaning
C++: PSIOString Visual Basic: string Java: String	A string containing the description for this Business Interlink plug-in.

Example

In the following C++ example, IntObj is a pointer to a Business Interlink object.

```
PSIOString& intobj_desc = IntObj->GetDescription();
```

In the following Visual Basic example, IntObj is a pointer to a Business Interlink object.

```
Dim intobj_desc As String
intobj_desc = IntObj.Description
```

In the following Java example, intobj is a pointer to a Business Interlink object.

```
String intobj_desc = intobj.GetDescription();
```

In the case of where XML code is set as follows,

```
<general_info>
   <description>PSCustomer services</description>
   <version>1</version>
</general_info>
```

the description returned would be “PSCustomer services”.

GetInterfaceName, InterfaceName

Syntax

C++:

```
PSIOString& GetInterfaceName()
```

Visual Basic:

```
string InterfaceName
```

Java:

```
String GetInterfaceName()
```

Description

The C++ method **GetInterfaceName** and the Visual Basic property **InterfaceName** get the name of a Business Interlink definition. This name is created in the PeopleSoft Application Designer.

The classes are:

- C++: InterfaceObject
- Visual Basic: IpsIntObj
- Java: PSJInterfaceObject

Parameters

None.

Returns

A string containing the Business Interlink definition name.

Value	Meaning
C++: PSIOString Visual Basic, Java: string	A string containing the Business Interlink definition name.

Example

Within PeopleCode, you use the name of the Business Interlink definition to instantiate a Business Interlink object, as in the following PeopleCode:

```
&QE_RP_SRAALL_1 = GetBusinessInterlink(BUSINESSINTERLINK.QE_COST);
```

The following code within a Business Interlink plug-in would fill the IntName variable with the Business Interlink definition name of “QE_COST”. IntObj is a pointer to the Business Interlink object.

C++:

```
PSIOString IntName;
IntName = IntObj->GetInterfaceName();
```

Visual Basic:

```
Dim IntName As String
IntName = IntObj.InterfaceName
```

Java:

```
String IntName = intobj.GetInterfaceName();
```

GetConfigParam, ConfigParam, GetConfigParamCount

Syntax

C++:

```
PSIOString& GetConfigParam(PSIOString&);
```

Visual Basic:

```
string ConfigParam
```

Java:

```
PSJVarInfo GetConfigParam(int index)
int GetConfigParamCount()
```

Description

The method **GetConfigParam** and the property **ConfigParam**, when given the name of a configuration parameter in this Business Interlink object, gets the value of one configuration parameter. This is used when the Interlink object only contains one configuration parameter. For Java, the method **GetConfigParamCount** gets the number of configuration parameters.

The classes are:

- C++: InterfaceObject
- Visual Basic: IPsIntObj
- Java: PSJInterfaceObject

Parameters

Parameter	Description
strParamName	Input. A PSIOString variable containing the name of the configuration parameter. This name is set in the XML design-time plug-in.

Returns

A string containing the configuration parameter value.

Value	Meaning
PSIOString&: C++ string: Visual Basic, Java	A string containing the value of the configuration parameter named in the strParamName input parameter.

Example

In the following examples, IntObj is a pointer to a Business Interlink object.

The following C++ code retrieves the value of the configuration parameter named “ConfigParameter”.

```
const PSIOString& value = IntObj->GetConfigParam("ConfigParameter");
```

The following Visual Basic code retrieves the value of the configuration parameter named “ConfigParameter”.

```
Dim ConfigName As String
ConfigName = IntObj.InterfaceName
```

The following Java code will set indexes pointing to the two configuration parameters named config1 and config2.

```
int nCountOut = 0;
nCountOut = intobj.GetConfigParamCount();
for (int i=0; i<nCountOut; i++) {
   try
   {
      PSJVarInfo varinfo = intobj.GetConfigParam(i);
      if(varinfo.strName.equals("config1"))
         int index_config1 = i;
      else if(varinfo.strName.equals("config2"))
         int index_config2 = i;
   }
   catch ( NoObjectReferenceException e)
   {
      return PSReturnStatus.FAILED;
   }
```

GetConfigParams

Syntax

C++:

```
VARINFOLIST& GetConfigParams()
```

Visual Basic:

```
GetConfigParams() As IPsEnumVarInfo
```

Description

The method **GetConfigParams** gets the information about the configuration parameters for a Business Interlink object—name, data type, default value, required—and places it into a list of type VARINFOLIST/PsEnumVarInfo. The configuration parameters are created in the XML design-time plug-in.

The classes are:

- C++: InterfaceObject
- Visual Basic: IPsIntObj

Parameters

None.

Returns

A list of type VARINFOLIST in C++, and PSEnumVarInfo in Visual Basic.

See Chapter 16, “Using The Business Interlink Methods,” Using Parameter Lists, page 245.

Example

In the following examples, IntObj is a pointer to a Business Interlink object.

The following C++ code retrieves the configuration parameter information and stores it in a VARINFOLIST variable.

```
VARINFOLIST varList = IntObj->GetConfigParams();
```

The following Visual Basic code retrieves the configuration parameter information and stores it in a VarInfo list named objConfigParams.

```
Set objConfigParams = pIntObj.GetConfigParams
```

In C++, after you use the GetConfigParams method to get the configuration parameters, use the following methods to retrieve the information about the parameters:

- c_str if your software system uses TCHAR data.
- c_astr if your software system uses char data.

For example, to get the names of the configuration parameters using Unicode data, use the c_str() method and m_strName (part of the VARINFOLIST parameter list structure).

```
TCHAR *config_name1 = (TCHAR *)varList[0]->m_strName.c_str();
TCHAR *config_name2 = (TCHAR *)varList[1]->m_strName.c_str();
```

After you use the Visual Basic **GetConfigParams** method to get the configuration parameters, you access the parameter information, as in the following example.

```
Dim objVarInfo As VarInfo
Dim objConfigParams As PsEnumVarInfo

Set objConfigParams = pIntObj.GetConfigParams
For Each objVarInfo In objConfigParams
```

```
      If objVarInfo.Name = "config1" Then
         string Configdata1 = objVarInfo.Data
      End If
      If objVarInfo.Name = "config2" Then
         string Configdata2 = objVarInfo.Data
      End If
   Next
```

GetGroup

Syntax

C++:

```
CriteriaNodeList* GetGroup()
```

Description

The method **GetGroup** gets a CriteriaNodeList structure. This structure will contain the grouping for the criteria rows for a query or update.

The class is CriteriaGroup.

Parameters

None.

Returns

CriteriaNodeList. This structure consists of pointers and indexes that form the groupings of the criteria rows for a query or update.

Example

The following code shows a call to a routine named GenerateCriteria, which has two inputs: the address of the first CriteriaNodeList structure, and a call to GetRows, which in this case, returns a pointer to the first CriteriaRowList structure. Then the GenerateCriteria routine uses the address to the first CriteriaNodeList structure to call GetGroup, getting the first CriteriaNode within the CriteriaNodeList structure. m_CriteriaGroup is a class of type CriteriaGroup that contains the first CriteriaNodeList for a Business Interlink object.

```
   GenerateCriteria(strCriteria, &IntObj->m_CriteriaGroup,
      IntObj->m_CriteriaGroup.GetRows());

   void RecordDriver::GenerateCriteria(PSIOString& strCriteria,
      CriteriaGroup* Group, CriteriaRowList *rowList)
   {
      CriteriaNodeList *gList = Group->GetGroup();
```

GetInputDocs

Syntax

C++:

```
CBIDocs& GetInputDocs(const TCHAR*)
```

Visual Basic:
```
GetInputDocs() As IPsBIDocs
```

Java:
```
PSJBIDocs GetInputDocs(String)
```

Description

The **GetInputDocs** method gets the top input document at the root level for a Business Interlink object. This is a hierarchical structure containing the values for the inputs for this Business Interlink object.

The classes are:

- C++: InterfaceObject
- Visual Basic: IpsIntObj
- Java: PSJInterfaceObject

Parameters

None. (C++ and Java pass in an empty string.)

Returns

The input document.

Value	Meaning
CBIDocs&, PsBIDocs, PSJBIDocs	The input document for a Business Interlink object. It contains the values for the input parameters.

Example

In the following C++ example, the CBIDocs input document for Calculate Cost, or the root level document, is created with the **GetInputDocs** method.

```
// Get the root level (Calculate Cost) CBIDocs input document
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

/* You can now get values (GetValue) and documents (GetNextDoc) from
this CBIDOcs input document */
```

In the following Visual Basic example, the PsBIDocs input document for Calculate Cost, or the root level document, is created with the **GetInputDocs** method.

```
' Get the root level (Calculate Cost) PsBIDocs input document
Set objInputDocs = pIntObj.GetInputDocs
objInputDocs.ResetCursor

' You can now get values (GetValue) and documents (GetNextDoc) from
' this PsBIDOcs input document */
```

In the following Java example, the PSJBIDocs input document for Calculate Cost, or the root level document, is created with the **GetInputDocs** method.

```
PSJBIDocs docsIn = intobj.GetInputDocs("");
docsIn.ResetCursor();
// You can now get values (GetValue) and documents (GetNextDoc) from
// this PSJBIDOcs input document */
// We explicitly call destroy to cleanup C++ memory behind the
// Java Doc objects which were created during Java processing.
docsIn.destroy();
```

GetInputParams, GetInputParam, GetInputParamCount

Syntax

C++:

```
VARINFOLIST& GetInputParams()
```

Visual Basic:

```
GetInputParams() As IPsEnumVarInfo
```

Java:

```
PSJVarInfo GetInputParam(int index)
int GetInputParamCount()
```

Description

The method **GetInputParams** and **GetInputParam** gets the information about the input parameters for a Business Interlink object—name, data type, default value, required—and places it into a list of type VARINFOLIST/PsEnumVarInfo. The input parameters for a transaction Business Interlink object are created in the XML design-time plug-in; for a Business Interlink object of type object update, object add, or object delete, they are selected from members of a class. For Java, the method **GetInputParamCount** gets the number of input parameters because the Java method GetInputParam gets one input parameter at a time.

The classes are:

- C++: InterfaceObject
- Visual Basic: IPsIntObj
- Java: PSJInterfaceObject

Parameters

Parameter	Description
index	For GetInputParams or GetInputParamCount: The location of the input parameter in the parameter list. Its value can be 0 to (number of parameters - 1).
none	For GetInputParam, no parameters.

Returns

A list of type VARINFOLIST in C++, and IPsEnumVarInfo in Visual Basic. In Java, for GetInputParam, one input parameter of type PSJVarInfo is returned, and for GetInputParamCount, an integer containing the number of parameters is returned.

See Chapter 16, “Using The Business Interlink Methods,” Using Parameter Lists, page 245.

Example

In the following examples, IntObj is a pointer to a Business Interlink object.

The following code retrieves the input parameter information and stores it in a VARINFOLIST named varList.

```
VARINFOLIST varList = IntObj->GetInputParams();
```

If the XML design-time plug-in for this Business Interlink object contains the following code,

```
<transaction name="transaction1">
   <input_list>
      <input name="input1"  type="string" required="true"/>
      <input name="input2"  type="string" required="false"/>
      <input name="input3"  type="int" required="true"/>
   </input_list>
   <output_list>
      <input name="output1"  type="string"/>
      <input name="output2"  type="string"/>
      <input name="output3"  type="int"/>
   </output_list>
</transaction>
```

then the following C++ code will set indexes pointing to each input based upon the names of the input parameters.

```
if(IntObj->GetObjName() == "transaction1")
{
   int index_input1, index_input2, index_input3;
   VARINFOLIST varListInput = IntObj->GetInputParams();
   for(int i = 0; i < varListInput.size(); i++)
   {
      if(varListInput[i]->m_strName == "input1")
         index_input1 = i;
      else if(varListInput[i]->m_strName == "input2")
         index_input2 = i;
      else if(varListInput[i]->m_strName == "input3")
         index_input3 = i;
   }
}
```

In Visual Basic, the following code will set indexes pointing to each input based upon the names of the input parameters.

```
lIndex = 0
index_input1 = -1
index_input2 = -1
index_input3 = -1

Set objInputParams = pIntObj.GetInputParams
For Each objVarInfo In objInputParams
   If objVarInfo.Name = "input1" Then
      index_input1 = lIndex
   ElseIf objVarInfo.Name = "input2" Then
      index_input2 = lIndex
   ElseIf objVarInfo.Name = "input3" Then
```

```
            index_input3 = lIndex
        End If
        lIndex = lIndex + 1
    Next
```

The following Java code will set indexes pointing to each input based upon the names of the input parameters.

```
if(intobj.GetObjName().equals("transaction1"))
{
   int nCountOut = 0;
   nCountOut = intobj.GetInputParamCount();
   for (int i=0; i<nCountOut; i++) {
      try
      {
         PSJVarInfo varinfo = intobj.GetInputParam(i);
         if(varinfo.strName.equals("input1"))
            int index_input1 = i;
         else if(varinfo.strName.equals("input2"))
            int index_input2 = i;
         else if(varinfo.strName.equals("input3"))
            int index_input3 = i;
      }
      catch ( NoObjectReferenceException e)
      {
         return PSReturnStatus.FAILED;
      }
   }
}
```

GetInterfaceType

Syntax

C++:

```
EIOCINTERFACESUPPORTED GetInterfaceType()
```

Visual Basic:

```
string InterfaceType
```

Java:

```
int GetInterfaceType()
```

Description

The **GetInterfaceType** method and the **InterfaceType** property get the type of this Business Interlink object. Use this method to determine what type this Business Interlink object is: transaction, object query, object add, object update, or object delete. If your Business Interlink plug-in only supports one type of Business Interlink object, your Business Interlink plug-in will not need to call this method.

The classes are:

- C++: InterfaceObject
- Visual Basic: IPsIntObj

• Java: PSJInterfaceObject

Parameters

None.

Returns

The Business Interlink object type.

Value	Meaning
EIOCINTERFACESUPPORTED	The type of the Business Interlink object. This can be: IOC_TRANSACTION (transaction)

Example

The following C++ code uses the GetInterfaceType method to determine which Business Interlink object execution method to call. IntObj is a pointer to a Business Interlink object.

```
EIOCEXECSTATUS eRt = INTOBJ_SUCCEED;
switch (IntObj->GetInterfaceType())
{
   case IOC_TRANSACTION:
      eRt = ExecuteTransaction(IntObj, nBatchMode);
      break;
         break;
   default:
      eRt = INTOBJ_FAILED;
}
```

GetObjName, ObjName

Syntax

C++:
```
PSIOString& GetObjName()
```

Visual Basic:
```
string ObjName
```

Java:
```
int GetObjName()
```

Description

The C++ **GetObjName** method and the Visual Basic property **ObjName** get the name of the Business Interlink object. The name of the Business Interlink object is creates in the XML design-time plug-in.

The classe are:

• C++: InterfaceObject

- Visual Basic: IPsIntObj
- Java: PSJInterfaceObject

Parameters

None.

Returns

A string containing the Business Interlink object name.

Value	Meaning
C++: PSIOString& Visual Basic: string Java: String	A string containing the Business Interlink object name.

Example

The following example uses the **GetObjName** method to see if the IntObj Business Interlink object is named “transaction1”.

```
if(IntObj->GetObjName() == _T("transaction1")
   { /* send an email message */ }
```

For example, if the XML design-time plug-in uses the following code to create a transaction Business Interlink object,

```
<transaction name="transaction1">
   <input_list>
      <input name="input1"  type="string" required="true"/>
      <input name="input2"  type="string" required="false"/>
      <input name="input3"  type="int" required="true"/>
   </input_list>
   <output_list>
      <input name="output1"  type="string"/>
      <input name="output2"  type="string"/>
      <input name="output3"  type="int"/>
   </output_list>
</transaction>
```

then the following C++ code will check for that transaction Business Interlink object name.

```
if(IntObj->GetObjName() == _T("transaction1")
{ /* Perform processing for transaction1 */ }
```

The following Visual Basic code will check for that transaction Business Interlink object name.

```
If pIntObj.ObjName = "transaction1" Then
   ’ Perform processing for transaction1
EndIf
```

The following Java code will check for that transaction Business Interlink object name.

```
if (intobj.GetObjName().equals("transaction1"))
```

```
{ /* Perform processing for transaction1 */ }
```

GetOutputDocs

Syntax

C++:
```
CBIDocs& GetOutputDocs(const TCHAR*)
```

Visual Basic:
```
GetOutputDocs() As IPsBIDocs
```

Java:
```
PSJBIDocs GetOutputDocs(String)
```

Description

The **GetOutputDocs** method gets the top output document at the root level for a Business Interlink object. This is a hierarchical structure that will contain the values for the outputs for this Business Interlink object.

The classes are:

- C++: InterfaceObject
- Visual Basic: IPsIntObj
- Java: PSJInterfaceObject

Parameters

None. (C++ and Java pass in an empty string.)

Returns

The output document.

Value	Meaning
CBIDocs&, IPsBIDocs, PSJBIDocs	The output document for a Business Interlink object. It will contain the values for the output parameters.

Example

In the following C++ example, the CBIDocs Output document for Calculate Cost, or the root level document, is created with the **GetOutputDocs** method.

```
// Get the root level (Calculate Cost) CBIDocs output document
CBIDocs docsOut = IntObj->GetOutputDocs(_T(""));
docsOut.Clear();

/* Get the output document names, and the member names, using GetOutputParams?
 (code not shown) */

/* You can now add values (AddValue) and documents (AddNextDoc) to
this CBIDocs output document */
```

In the following Visual Basic example, the PsBIDocs output document for Calculate Cost, or the root level document, is created with the **GetOutputDocs** method.

```
' Get the root level (Calculate Cost) PsBIDocs output document
Set objOutputDocs = pIntObj.GetOutputDocs
objOutputDocs.Clear

' You can now add values (AddValue) and documents (AddNextDoc) to
' this PsBIDocs output document.
```

In the following Java example, the PSBIDocs output document for Calculate Cost, or the root level document, is created with the **GetOutputDocs** method.

```
PSJBIDocs docsOut = intobj.GetOutputDocs("");
docsOut.Clear();
// You can now add values (AddValue) and documents (AddNextDoc) to
// this PsBIDocs output document.
// We explicitly call destroy to cleanup C++ memory behind the
// Java Doc objects which were created during Java processing.
docsOut.destroy();
```

GetOutputParams, GetOutputParam, GetOutputParamCount

Syntax

C++:

```
VARINFOLIST& GetOutputParams()
```

Visual Basic:

```
GetOutputParams() As IPsEnumVarInfo
```

Java:

```
PSJVarInfo GetOutputParam(int index)
int GetOutputParamCount()
```

Description

The methods **GetOutputParams** and **GetOutputParam** get the information about the output parameters for a Business Interlink object—name, data type, default value, required—and places it into a parameter list. The output parameters for a transaction Business Interlink object are created in the XML design-time plug-in; for a Business Interlink object of type object query, they are selected from data members of a class. For Java, the method **GetOutputParamCount** gets the number of output parameters because the Java method GetOutputParam gets one output parameter at a time.

The classes are:

- C++: InterfaceObject
- Visual Basic: IpsIntObj
- Java: PSJInterfaceObject

Parameters

Parameter	Description
index	For GetOutputParam: The location of the output parameter in the parameter list. Its value can be 0 to (number of parameters - 1).
None.	GetOutputParams and GetOutputParamCount have no parameters.

Returns

A list of type VARINFOLIST in C++, and IPsEnumVarInfo in Visual Basic. In Java, for GetOutputParam, one output parameter of type PSJVarInfo is returned, and for GetOutputParamCount, an integer containing the number of parameters is returned.

See Chapter 16, "Using The Business Interlink Methods," Using Parameter Lists, page 245.

Example

In the following examples, IntObj is a pointer to a Business Interlink object.

The following code retrieves the input parameter information and stores it in a VARINFOLIST named varListOutput.

```
VARINFOLIST varListOutput = IntObj->GetOutputParams();
```

For example, if the XML design time plug-in uses the following code to create the output parameters,

```
<transaction name="transaction1">
   <input_list>
      <input name="input1"  type="string" required="true"/>
      <input name="input2"  type="string" required="false"/>
      <input name="input3"  type="int" required="true"/>
   </input_list>
   <output_list>
      <input name="output1"  type="string"/>
      <input name="output2"  type="string"/>
      <input name="output3"  type="int"/>
   </output_list>
</transaction>
```

then the following C++ code will set indexes pointing to each output based upon the names of the output parameters.

```
if(IntObj->GetObjName() == "transaction1")
{
   int index_output1, index_output2, index_output3;
  VARINFOLIST varListInput = IntObj->GetOutputParams();
   for(int i = 0; i < varListOutput.size(); i++)
   {
      if(varListOutput [i]->m_strName == "output1")
         index_output1 = i;
      else if(varListOutput [i]->m_strName == "output2")
         index_output2 = i;
      else if(varListOutput [i]->m_strName == "output3")
         index_output3 = i;
```

```
  }
}
```

and the following Visual Basic code will set indexes pointing to each output based upon the names of the output parameters.

```
Dim objVarInfo As VarInfo
Dim objOutputParams As PsEnumVarInfo
Dim lIndex As Long
Dim index_output1 As Long
Dim index_output2 As Long
Dim index_output3 As Long


Set objOutputParams = pIntObj.GetOutputParams
For Each objVarInfo In objOutputParams
   If objVarInfo.Name = "output1" Then
      index_output1 = lIndex
   ElseIf objVarInfo.Name = "output2" Then
      index_output2 = lIndex
   ElseIf objVarInfo.Name = "output3" Then
      index_output3 = lIndex
   End If
   lIndex = lIndex + 1
Next
```

and the following Java code will set indexes pointing to each output based upon the names of the output parameters.

```
if(intobj.GetObjName().equals("transaction1"))
{
   int nCountOut = 0;
   nCountOut = intobj.GetOutputParamCount();
   for (int i=0; i<nCountOut; i++) {
      try
      {
         PSJVarInfo varinfo = intobj.GetOutputParam(i);
         if(varinfo.strName.equal("output1"))
            int index_output1 = i;
         else if(varinfo.strName.equal("output2"))
            int index_output2 = i;
         else if(varinfo.strName.equal("output3"))
            int index_output3 = i;
      }
      catch ( NoObjectReferenceException e)
      {
         return PSReturnStatus.FAILED;
      }
   }
}
```

GetRows

Syntax

```
CriteriaRowList* GetRows();
```

Description

The method **GetRows** gets a CriteriaRowList structure. This structure will contain the criteria rows for a query or update.

The C++ class is CriteriaGroup.

Parameters

None.

Returns

A CriteriaRowList. The CriteriaRowList structure consists of a list of CriteriaRow structures. The CriteriaRowList and CriteriaRow structures are included with the query Business Interlink object that is input for the ExecuteObjectQuery method. The CriteriaRow structure consists of a criteria row, and has the following form.

```
int rowNum;
PSIOString lOper;      // logical operator
PSIOString lhsExpr;    // left hand side
PSIOString rOper;      // relational operator
PSIOString rhsExpr;    // right hand side
PSIOString rhsDefault; // default value of right hand side
PSIOString dataType;   // the data type
```

Example

The following code calls a routine named GenerateCriteria, which has two inputs: the address of the first CriteriaNodeList structure, and a call to GetRows, which in this case, returns a pointer to the first CriteriaRowList structure. m_CriteriaGroup is a class of type CriteriaGroup that contains the first CriteriaNodeList for a Business Interlink object.

```
GenerateCriteria(strCriteria, &IntObj->m_CriteriaGroup,
   IntObj->m_CriteriaGroup.GetRows());
```

InterfaceName

Description

Refer to GetInterfaceName for this method.

Writing a Runtime Plug-in With Plug-in Class Methods

You write the runtime plug-in using the runtime plug-in class methods. The class for these methods is the class for your Interlink object.

Plug-In Class Methods

This section describes the plug-in class methods, in alphabetical order.

ExecuteTransaction

Syntax

C++:

```
virtual EIOCEXECSTATUS ExecuteTransaction(InterfaceObject * IntObj, const int BatchMode);
```

Visual Basic:

```
ExecuteTransaction(IntObj As PsIntObj, BatchMode As Long) As ENUM_EIOCEXECSTATUS
```

Java:

```
int ExecuteTransaction(PSJInterfaceObject intobj)
```

Description

The ExecuteTransaction method is the method that executes the transaction runtime plug-in. You must write the code for this method when you are executing an Interlink object that is of type transaction.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver
- Java: PSJInterfaceObject

Parameters

Parameter	Description
IntObj	An Interlink object. This is provided by the Interlink Framework.
BatchMode	Not currently used.

Returns

The return status. EIOCEXECSTATUS for C++, ENUM_EIOCEXECSTATUS for Visual Basic, or int for Java.

Value	Meaning
INTOBJ_SUCCEED	Return this if the method succeeds.
INTOBJ_FAILED	Return this if the method fails.

GetVersion, PsIoDriver_GetVer

Syntax

C++:

```
virtual const TCHAR * GetVersion() = 0;
```

Visual Basic:

```
PsIoDriver_GetVer() As String
```

Java:

```
String GetVersion()
```

Description

Write this methods to return a string containing the version number of this Interface Definition. This string is used in the New Interface Definition page. Since GetVersion is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver
- Java: PSJInterfaceObject

Parameters

None.

Returns

A string containing the version number.

Value	Meaning
A character string.	Contains the version number of this Interface Definition.

Example

The following C++ example sets the version number to 1.00.

```
const TCHAR * SimpleDriver::GetVersion() const
{
   return _T("1.00");
}
```

The following Visual Basic example sets the version number to 1.0.0.

```
Private Function PsIoDriver_GetVer() As String
   '
   PsIoDriver_GetVer = "1.0.0"
   '
End Function
```

The following Java example sets the version number to 2.11.00.

```
public String GetVersion(){
   return "2.11.00"; }
```

IsVersionCompatible, PsIoDriver_IsVersionCompatible

Syntax

C++:

```
virtual BOOL IsVersionCompatible(TCHAR* version) = 0;
```

Visual Basic:

```
PsIoDriver_IsVersionCompatible(ByVal version As String) As Long
```

Java:

```
boolean IsVersionCompatible(String version)
```

Description

Write this method to return true or false: true if this plug-in version number is for a version that your plug-in supports, false otherwise. The Business Interlink Framework calls this method to see if this plug-in is compatible with other versions of this plug-in that are currently in the PeopleTools database. For example, this method tells if an older version of a plug-in is compatible with a new version. Since IsVersionCompatible is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver.
- Visual Basic: PsIoDriver.
- Java: PSJInterfaceObject.

Parameters

Parameter	Description
version	the plug-in version number that is passed in from other Interface Drivers in the PeopleTools database.

Returns

True if version is supported, false otherwise.

Value	Meaning
BOOL, Long	True if *version* is a supported type for your plug-in, False otherwise.

Example

The following C++ example returns TRUE, meaning that this plug-in is compatible with any older versions of this plug-in.

```
virtual BOOL IsVersionCompatible(const TCHAR* version) { return TRUE; }
```

The following Visual Basic example returns True, meaning that this plug-in is compatible with any older versions of this plug-in.

```
Private Function PsIoDriver_IsVersionCompatible(ByVal bstrVerion As String) As Long
    '
    PsIoDriver_IsVersionCompatible = True
    '
End Function
```

The following Java example returns True, meaning that this plug-in is compatible with any older versions of this plug-in.

```
public boolean IsVersionCompatible() {return true;}
```

Accessing Hierarchical Input/Output Values With BiDoc Class Methods

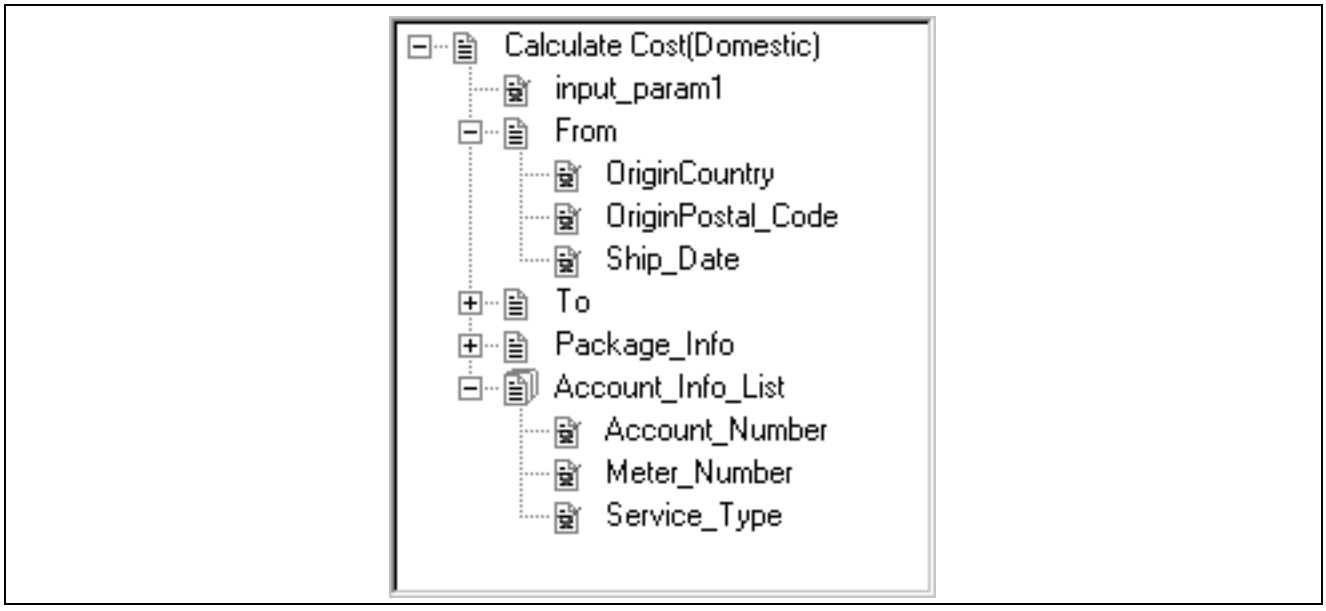
You use the CBIDocs/PsCBIDocs methods to access the inputs and output values of a Business Interlink object when the inputs and outputs are stored in a document. This is a hierarchical structure, as opposed to flat tables.

For C++, you must include the header file psbidoc.h for this class.

A Business Interlink can have hierarchical data, meaning that the inputs and outputs for a Business Interlink object are stored in the form of a CBIDocs document. Here is an example CBIDocs document: an edited version of the Calculate Cost(Domestic) transaction from the Freight Carrier plug-in.

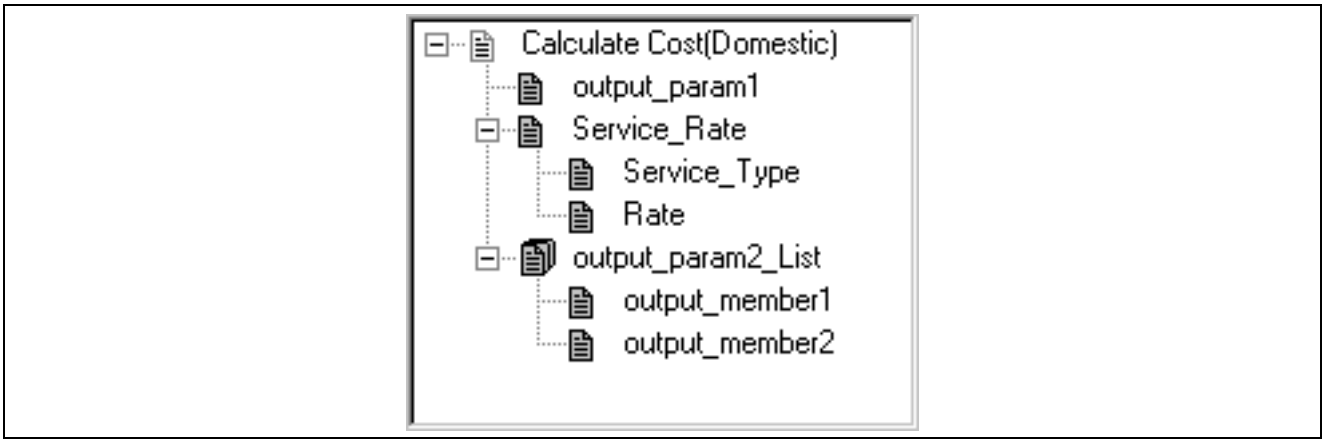
Note. To better illustrate the CBIDocs methods, this example was created from an edited version of the Freight Carrier plug-in. The input parameter input-param1 and output parameters output_param1 and output_param2_list were added to this example, and the Account_Info input parameter was modified to be a list.

Following is the input document.



Example CBIDocs Input Document

Following is the output document.



Example CBIDocs Output Document

When you have hierarchical data, use the CBIDocs methods to add input data and get output data from the CBIDocs document.

The **GetInputDocs** method accesses a CBIDocs input document at the root level; for the example above, this is the Calculate Cost(Domestic) input document. To get another CBIDocs input document, use the **GetNextDoc** or **GetPreviousDoc** methods. Since Business Classes, when they are input parameters, are stored as documents within the CBIDocs input document, use the **GetDoc** method to get them and their members; in the example, From, To, Package_Info, and Account_Info_List are stored as documents. Since documents can also be lists, such as Account_Info_List, use the **GetNextDoc** or **GetPreviousDoc** methods. Use the **GetValue** method to get values from either input parameters of basic type, such as input_param1 in the example, or to members of input parameters which are documents, such as OriginCountry in the From input document.

You can use the **GetCount** method to get the number of documents in an input document list, such as the number of Account_Info_List documents in the above example. To move to any document in the list without having to use **GetNextDoc** to cycle through several of them, use **MoveToDoc**.

The **GetOutputDocs** method accesses a CBIDocs output document at the root level; for the example above, this is the Calculate Cost(Domestic) output document. To add another CBIDocs output document, use the **AddNextDoc** method. Since Business Classes, when they are output parameters, are stored as documents within the CBIDocs output document, use the **AddDoc** method to add them; in the example, Service_Rate and output_param2_List are stored as documents. Since some documents can also be lists, such as output_param2_List in the example above, use the **AddNextDoc** method to add a document to the list. Use the **AddValue** method to add values to either output parameters of basic type, such as output_param1 in the example, or to members of output parameters that are documents, such as Service_Type in Service_Rate.

BIDoc Class Methods

This section describes the BIDoc class methods in alphabetical order.

AddDoc

Syntax

C++:

```
CBIDocs AddDoc(const TCHAR*)
```

Visual Basic:

```
AddDoc(String) As PsBIDocs
```

Java:

```
PSJBIDocs AddDoc(String)
```

Description

Adds a document to an output document. The added document is an output parameter for a Business Interlink object that is not of simple type (such as integer or string). You must add the document to the output document before you can add values to its members with **AddValue**.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

Parameter	Description
TCHAR*, string	The name of the document that AddDoc adds to the output document.

Returns

The document added to the output document.

Value	Meaning
CBIDocs, PsBIDocs, PSJBIDocs	The document that **AddDoc** adds to the output document.

Example

In the following example, the output document for Calculate Cost, or the root level document, is created with the **GetOutputDocs** method. The Calculate Cost output parameter Service_Rate is a document, so the **AddDoc** method adds it to the output document.

C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

CBIDocs docsOut = IntObj->GetOutputDocs(_T(""));
docsOut.Clear();

/* Get the output document names, and the member names, using GetOutputParams?
 (code not shown) */

while(docsIn.GetStatus() == eNoError) {

/* Get the input values and call the external system to get values to put into?
 the CBIDocs output documents (code not shown) */
```

```
    if (!docsOut.Empty()) {
        docsOut.AddNextDoc();
    }

    CBIDocs docRate = docsOut.AddDoc(_T("Service_Rate"));
    PSIOString rate;
    /* Get value for serviceType (code not shown) */
    docRate.AddValue(_T("Rate"), rate.c_str());

    /* Call GetValue for output_param1, call AddDoc, GetValue, and
       GetNextDoc for output_param2_List (code not shown) */

    docsIn.GetNextDoc();
} //end while
```

Visual Basic:

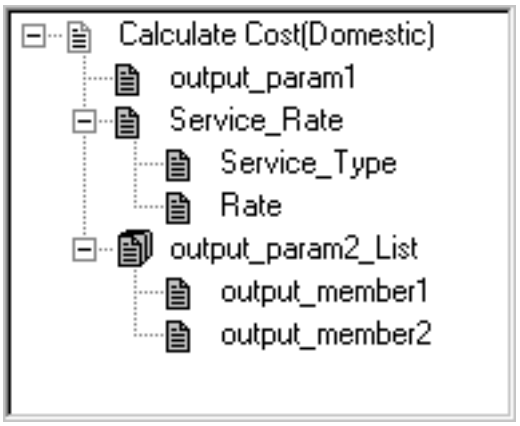
```
Dim objOutputDocs As PsBIDocs
Dim objInputDocs As PsBIDocs
Dim lResult As Long
Dim eNoError As Integer
Dim objDocRate As PsBIDocs
Dim lRate As String

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor

Set objOutputDocs = IntObj.GetOutputDocs
objOutputDocs.Clear

’ Get the output document names, and the member names, using
’ GetOutputParams (code not shown)

While objInputDocs.GetStatus = eNoError

’ Get the input values and call the external system to get values to
’ put into the CBIDocs output documents (code not shown)

   If objOutputDocs.Empty <> 1 Then
      lResult = objOutputDocs.AddNextDoc
   End If


   Set objDocRate = objOutputDocs.AddDoc("Service_Rate")
   ’ Get value for rate (code not shown)
   lResult = objDocRate.AddValue("Rate", lRate)

   ’ Call GetValue for output_param1, call AddDoc, GetValue, and
   ’ GetNextDoc for output_param2_List (code not shown)
```

```
      objInputDocs.GetNextDoc
   Wend
```

Java:

```
   PSJBIDocs docsIn = intobj.GetInputDocs("");
   docsIn.ResetCursor();

   PSJBIDocs docsOut = intobj.GetOutputDocs("");
   docsOut.Clear();

   // Get the output document names, and the member names, using GetOutputParams
   // (code not shown)

   // GetCount method, docsInStatus variable used instead of GetStatus
   int rootCount = docsIn.GetCount("root");
   int docsInStatus = -1;
   if (rootCount > 0) { docsInStatus = 0;}
   while(docsInStatus == 0) {

   // Get the input values and call the external system to get values to put into
   // the output documents (code not shown)

      docsOut.AddNextDoc();

      String serviceType = new String();
      String rate = new String();
      PSJBIDocs docRate = docsOut.AddDoc("Service_Rate");
      // Get value for serviceType (code not shown)
      docRate.AddValue("Rate", rate);
      // Call GetValue for output_param1, call AddDoc, GetValue, and
      // GetNextDoc for output_param2_List (code not shown)

      docsInStatus = docsIn.GetNextDoc();
   } //end while
   // Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
   docRate.destroy();
   docsOut.destroy();
   docsIn.destroy();
```

The figure below shows the output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Freight Carrier plug-in that was modified for this example (output_param1 and output_param2_List were added).



Example CBIDocs Output Document

AddNextDoc

Syntax

C++:

Class: CBIDocs

```
int AddNextDoc();
```

Visual Basic:

```
AddNextDoc() As Long
```

Java:

```
int AddNextDoc();
```

Description

The **AddNextDoc** method adds a document to one of the following levels:

- The root level of the output document for a Business Interlink object. This level was created with the method **GetInputDocs**.
- When a document within the output document is a list, **AddNextDoc** adds another document to the list. If you use **AddNextDoc** on a document that is not a list, **AddNextDoc** fails and returns an error.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

None.

Returns

An integer with the following possible values:

Number	Enum value and Meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

In the following example, the output document for Calculate Cost, or the root level document, is created with the **GetOutputDocs** method. **AddNextDoc** will add more root level (Calculate Cost) documents at this level.

The Calculate Cost output parameter output_param2_List is a document, so it is added with the **AddDoc** method. **AddNextDoc** will add more output_param2_List documents.

C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();
```

```
CBIDocs docsOut = IntObj->GetOutputDocs(_T(""));
docsOut.Clear();

/* Get the output document names, and the member names, using GetOutputParams?
 (code not shown) */

while(docsIn.GetStatus() == eNoError) {

/* Get the input values and call the external system to get values to put?
 into the CBIDocs output documents (code not shown) */

   if (!docsOut.Empty()) {
       docsOut.AddNextDoc();
   }

   PSIOString serviceType, rate;
   CBIDocs docRate = docsOut.AddDoc(_T("Service_Rate"));
   /* Get value for rate, serviceType (code not shown) */
   docRate.AddValue(_T("Service_Type"), serviceType.c_str());
   docRate.AddValue(_T("Rate"), rate.c_str());

// number_of_docOPList = number of docs in output_param2_List (code not shown)
   PSIOString postalcode[number_of_docOPList];
   PSIOString country[number_of_docOPList];
   CBIDocs docOPlist = docsOut.AddDoc(_T("output_param2_List"));
   /* Get values for value1 and value2 (code not shown) */
   for(int n = 0; n < number_of_docOPList; n++) {
      docOPlist.AddValue(_T("output_member1"),value1[n].c_str);
      docOPlist.AddValue(_T("output_member2"), value2[n].c_str));
      docOPlist.AddNextDoc();
   }

   docsIn.GetNextDoc();
} //end while
```

Visual Basic:

```
Dim objOutputDocs As PsBIDocs
Dim objInputDocs As PsBIDocs
Dim lResult As Long
Dim eNoError As Integer
Dim objDocRate As PsBIDocs
Dim lRate As String
Dim lServiceType As String
Dim lValue As String

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor

Set objOutputDocs = IntObj.GetOutputDocs
```

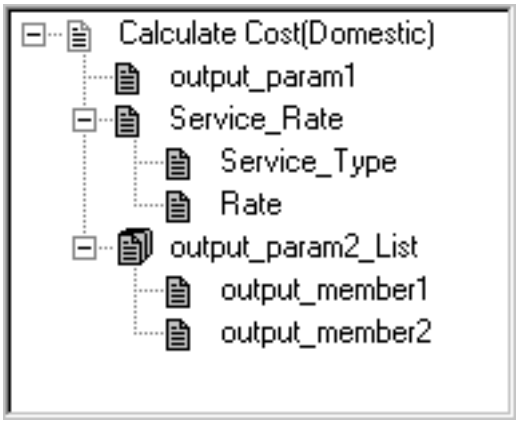
```
objOutputDocs.Clear

' Get the output document names, and the member names, using
' GetOutputParams (code not shown)

While objInputDocs.GetStatus = eNoError

' Get the input values and call the external system to get values to
' put into the CBIDocs output documents (code not shown)

   If objOutputDocs.Empty <> 1 Then
      lResult = objOutputDocs.AddNextDoc
   End If

   Set objDocRate = objOutputDocs.AddDoc("Service_Rate")
   ' Get value for rate (code not shown)
   lResult = objDocRate.AddValue("Rate", lRate)
   lResult = objDocRate.AddValue("Service_Type", lServiceType)
   Set objDocOPlist = objOutputDocs.AddDoc("output_param2_List")

' number_of_docOPList = number of docs in output_param2_List (code not shown)
   For lIndex = 1 To number_of_docOPList
      lResult = objDocOPlist.AddValue("output_member1",
         value1[lIndex])
      lResult = objDocOPlist.AddValue("output_member2",
         value2[lIndex])
      objOutputDocs.AddNextDoc()
   Next

   objInputDocs.GetNextDoc
Wend
```

Java:

```
PSJBIDocs docsIn = intobj.GetInputDocs("");
docsIn.ResetCursor();

PSJBIDocs docsOut = intobj.GetOutputDocs("");
docsOut.Clear();

// Get the output document names, and the member names, using GetOutputParams
// (code not shown)

// GetCount method, docsInStatus variable used instead of GetStatus
int rootCount = docsIn.GetCount("root");
int docsInStatus = -1;
if (rootCount > 0) { docsInStatus = 0;}
while(docsInStatus == 0) {

// Get the input values and call the external system to get values to put into
// the output documents (code not shown)
```

```
      docsOut.AddNextDoc();

      String serviceType = new String();
      String rate = new String();
      PSJBIDocs docRate = docsOut.AddDoc("Service_Rate");
      // Get value for rate, serviceType (code not shown)
      docRate.AddValue("Service_Type", serviceType);
      docRate.AddValue("Rate", rate);

   // number_of_docOPList = number of docs in output_param2_List (code not shown)
      String[] postalcode = new String[number_of_docOPList];
      String[] country = new String [number_of_docOPList];
      PSJBIDocs docOPlist = docsOut.AddDoc("output_param2_List");
      // Get values for value1 and value2 (code not shown)
      for(int n = 0; n < number_of_docOPList; n++) {
         docOPlist.AddValue("output_member1",value1[n]);
         docOPlist.AddValue("output_member2",value2[n]);
         docOPlist.AddNextDoc();
      }

      docsInStatus = docsIn.GetNextDoc();
   } //end while
   // Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
   docRate.destroy();
   docsOut.destroy();
   docsIn.destroy();
```

The figure below shows the output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Freight Carrier plug-in that was modified for this example (output_param1 and output_param2_List were added).



Example CBIDocs Output Document

AddValue, AddValueInt, AddValueFloat, AddValueDouble

Syntax

C++:

```
int AddValue(const TCHAR* name, const TCHAR* value);
int AddValue(const TCHAR* name, const int value);
int AddValue(const TCHAR* name, const double value);
int AddValue(const TCHAR* name, const float value);
```

Visual Basic:

```
AddValue(name As String, value As String) As Long
AddValueInt(name As String, value As Int) As Long
AddValueFloat(name As String, value As Float) As Long
AddValueDouble(name As String, value As Double) As Long
```

Java:

```
AddValue(String name, String value);
int AddValue(String name, Integer value);
int AddValue(String name, Float value);
int AddValue(String name, Double value);
```

Description

Adds a value to a member of a document within the output document for a Business Interlink object.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

Parameter	Description
name	The name of the member of the document that is having a value added to it.
value	The value that is added.

Returns

An integer with the following possible values:

Number	Enum value and Meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.

Number	Enum value and Meaning
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

In the following example, the output document for Calculate Cost, or the root level document, is created with the **GetOutputDocs** method. **AddValue** adds a value to the output parameter output_param1. The Calculate Cost output parameter Service_Rate is a document, so the **AddDoc** method adds it to the CBIDocs output document. Then the **AddValue** method adds values to the Service_Rate document members.

C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

CBIDocs docsOut = IntObj->GetOutputDocs(_T(""));
docsOut.Clear();

/* Get the output document names, and the member names, using GetOutputParams?
 (code not shown) */
```

```
while(docsIn.GetStatus() == eNoError) {

/* Get the input values and call the external system to get values to put?
 into the CBIDocs output documents (code not shown) */

   if (!docsOut.Empty()) {
       docsOut.AddNextDoc();
   }

   docsOut.AddValue(_T("output_param1"), _T("string_value"));

   CBIDocs docRate = docsOut.AddDoc(_T("Service_Rate"));
   docRate.AddValue(_T("Service_Type"), serviceType.c_str());
   docRate.AddValue(_T("Rate"), rate.c_str());

   /* Call AddDoc, AddValue, and AddNextDoc for output_param2_List
      (code not shown) */

   docsIn.GetNextDoc();
} //end while
```

Visual Basic:

```
Dim objOutputDocs As PsBIDocs
Dim objInputDocs As PsBIDocs
Dim lResult As Long
Dim eNoError As Integer
Dim objDocRate As PsBIDocs
Dim lRate As String
Dim lServiceType As String
Dim lValue As String

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor

Set objOutputDocs = IntObj.GetOutputDocs
objOutputDocs.Clear

’ Get the output document names, and the member names, using
’ GetOutputParams (code not shown)

While objInputDocs.GetStatus = eNoError

’ Get the input values and call the external system to get values to
’ put into the CBIDocs output documents (code not shown)

   If objOutputDocs.Empty <> 1 Then
      lResult = objOutputDocs.AddNextDoc
   End If
```

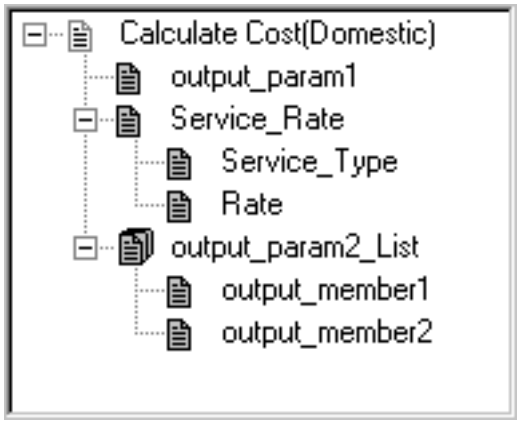
```
   lResult = objOutputDocs.AddValue("output_param1", lValue)

   Set objDocRate = objOutputDocs.AddDoc("Service_Rate")
   ' Get value for rate (code not shown)
   lResult = objDocRate.AddValue("Rate", lRate)
   lResult = objDocRate.AddValue("Service_Type", lServiceType)
   Set objDocOPlist = objOutputDocs.AddDoc("output_param2_List")

   ' Call AddDoc, AddValue, and AddNextDoc for output_param2_List
   '  (code not shown) */

   objInputDocs.GetNextDoc
Wend
```

Java:

```
PSJBIDocs docsIn = IntObj.GetInputDocs("");
docsIn.ResetCursor();

try {

   PSJBIDocs docsOut = IntObj.GetOutputDocs("");
   docsOut.Clear();

   // Get the output document names, and the member names, using GetOutputParams
   // (code not shown)

   // GetCount method, docsInStatus variable used instead of GetStatus
   int rootCount = docsIn.GetCount("root");
   int docsInStatus = -1;
   if (rootCount > 0) { docsInStatus = 0;}
   while(docsInStatus == 0) {

   // Get the input values and call the external system to get values to put
   // into the CBIDocs output documents (code not shown)

      docsOut.AddNextDoc();

      docsOut.AddValue("output_param1", "string_value");

      PSJBIDocs docRate = docsOut.AddDoc("Service_Rate");
      docRate.AddValue("Service_Type", serviceType);
      docRate.AddValue("Rate", rate);
   }

/* Call AddDoc, AddValue, and AddNextDoc for output_param2_List
         (code not shown) */

      docsInStatus = docsIn.GetNextDoc();
   } //end while
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
```

```
docRate.destroy();
docsOut.destroy();
docsIn.destroy();
```

The figure below shows the output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Freight Carrier plug-in that was modified for this example (output_param1 and output_param2_List were added).



Example CBIDocs Output Document

Clear

Syntax

C++:

```
void Clear()
```

Visual Basic:

```
Clear()
```

Java:

```
Clear()
```

Description

Returns the document for a Business Interlink object to its initial state, containing no values. The Clear method is used on the output document in order to clear it before adding documents and values to it. Once you clear it, you may need to use the **GetOutputParams** method to get the names of the output parameters.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

None.

Returns

None.

Example

The following example uses the **Clear** method to clear docsOut, which is an output document for a Business Interlink object.

C++:

```
CBIDocs docsOut = IntObj->GetOutputDocs(_T(""));
docsOut.Clear();
```

Visual Basic:

```
Set docsOut = IntObj.GetOutputDocs
docsOut.Clear
```

Java:

```
PSJBIDocs docsOut = intobj.GetOutputDocs("");
docsOut.Clear();
```

destroy

Description

Used only with Java. Performs memory cleanup for the GetInputDocs, GetOutputDocs, GetDoc, and AddDoc methods. Adds a document to an output document. Call destroy once for each time you call these methods. The destroy method cleans up C++ memory behind the Java Doc objects which was created during Java processing.

The class is PSJBIDocs for Java.

Parameters

None.

Returns

None.

Example

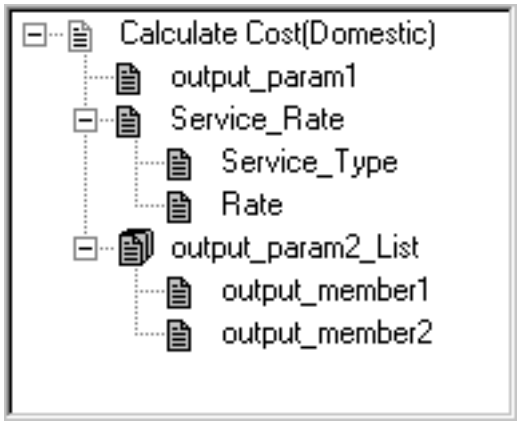
In the following example, the output document for Calculate Cost, or the root level document, is created with the **GetOutputDocs** method. The Calculate Cost output parameter Service_Rate is a document, so the **AddDoc** method adds it to the output document.

Java:

```
PSJBIDocs docsIn = intobj.GetInputDocs("");
docsIn.ResetCursor();

PSJBIDocs docsOut = intobj.GetOutputDocs("");
docsOut.Clear();

// Get the output document names, and the member names, using GetOutputParams
// (code not shown)

// GetCount method, docsInStatus variable used instead of GetStatus
int rootCount = docsIn.GetCount("root");
int docsInStatus = -1;
```

```
if (rootCount > 0) { docsInStatus = 0;}
while(docsInStatus == 0) {

// Get the input values and call the external system to get values to put into
// the output documents (code not shown)

   docsOut.AddNextDoc();

   String serviceType = new String();
   String rate = new String();
   PSJBIDocs docRate = docsOut.AddDoc("Service_Rate");
   // Get value for serviceType (code not shown)
   docRate.AddValue("Rate", rate);
  // Call GetValue for output_param1, call AddDoc, GetValue, and
   // GetNextDoc for output_param2_List (code not shown)

   docsInStatus = docsIn.GetNextDoc();
} //end while
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
docRate.destroy();
docsOut.destroy();
docsIn.destroy();
```

The figure below shows the output document for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Freight Carrier plug-in that was modified for this example (output_param1 and output_param2_List were added).



Example CBIDocs Output Document

Empty

Syntax

C++:

```
bool Empty();
```

Visual Basic:

```
Empty() As Long
```

Description

Tests the document for a Business Interlink object to see if it is empty. This is often done for the output document before calling **AddNextDoc** for the first time.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs

Parameters

Parameter	Description
None.	

Returns

Whether or not the document is empty.

Value	Meaning
0	The document is not empty (it exists).
1	The document is empty (it does not exist).

Example

The following example uses the **Empty** method to test docsOut, which is an output document for a Business Interlink object, to see if it exists. If it does, the **AddNextDoc** method adds an output document.

C++:

```
if(!docsOut.Empty())
   docsOut.AddNextDoc();
```

Visual Basic:

```
If docsOut.Empty <> 1 Then
   lResult = docsOut.AddNextDoc
End If
```

GetCount

Syntax

C++:

```
int GetCount(const TCHAR* name);
int GetCount(const PSIOString & name);
```

Visual Basic:

```
GetCount(name As String) As Long
```

Java:

```
int GetCount(String name);
```

Description

Returns the number of documents within a document list or parameter list contained within a CBIDocs input document for a Business Interlink object.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

Parameter	Description
name	The name of the document list or parameter list.

Returns

The number of documents in the list.

Value	Meaning
Int, Long	The number of documents in the list.

Example

In the following example, the input document for Calculate Cost, or the root level document, is created with the **GetInputDocs** method. The Calculate Cost input parameter Account_Info_List is a document list, so the **GetDoc** method gets it. The **GetCount** document gets the number of Account_Info_List documents in the list, and **GetNextDoc** gets each document from the list.

C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

/* Call GetValue for input_param1, GetDoc, GetValue for From, To, Package_Info?
 (code not shown) */

while(docsIn.GetStatus() == eNoError) {

   CBIDocs docAccountInfo = docsIn.GetDoc(_T("Account_Info_List"));
   int count = docsIn.GetCount(_T("Account_Info_List"));
   for(int i = 0; i < count; i++) {
      str1 = docAccountInfo.GetValue(_T("Account_Number"));
      str2 = docAccountInfo.GetValue(_T("Meter_Number"));
      const TCHAR *pStr = docAccountInfo.GetValue(_T("Service_Type"));
      docAccountInfo.GetNextDoc();
   }
```

```
/* Call the external system and put the resulting values into the CBIDocs output?
 documents (code not shown) */

   docsIn.GetNextDoc();
} //end while
```

Visual Basic:

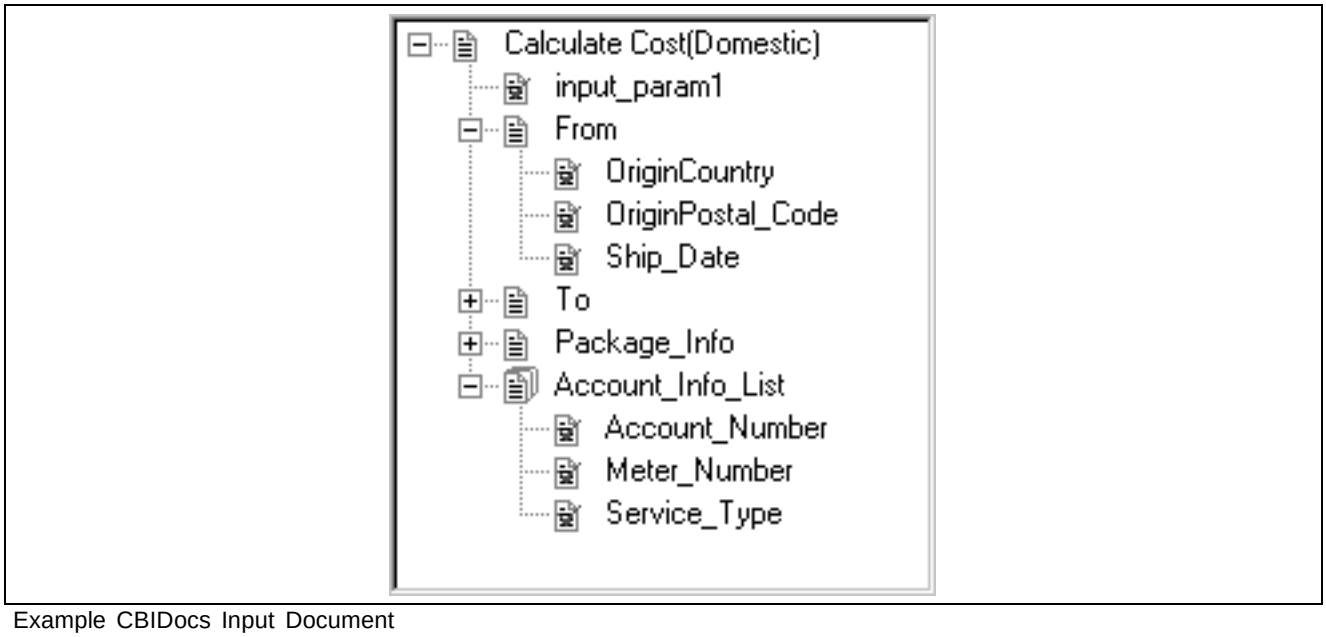
```
Dim str1 As String
Dim str2 As String
Dim count As Long
Dim lIndex As Long
Dim objInputDocs As PsBIDocs
Dim objDocAccountInfo As PsBIDocs

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor

' Call GetValue for input_param1, GetDoc, GetValue for From, To,
' Package_Info (code not shown) */

While objInputDocs.GetStatus = eNoError

   Set objDocAccountInfo = objInputDocs.GetDoc("Account_Info_List")
   count = objInputDocs.GetCount("Account_Info_List")
   For lIndex = 1 To count
      str1 = objDocAccountInfo.GetValue("Account_Number")
      str2 = objDocAccountInfo.GetValue("Meter_Number")
      str3 = objDocAccountInfo.GetValue("Meter_Number")
      objDocAccountInfo.GetNextDoc
   Next

' Call the external system and put the resulting values into the
' output documents (code not shown)

   objInputDocs.GetNextDoc
Wend
```

Java:

```
PSJBIDocs docsIn = intobj.GetInputDocs("");
docsIn.ResetCursor();
String str1 = new String();
String str2 = new String();
String str3 = new String();

// Call GetValue for input_param1, GetDoc, GetValue for From, To, Package_Info
// (code not shown)

// GetCount method, docsInStatus variable used instead of GetStatus
int rootCount = docsIn.GetCount("root");
int docsInStatus = -1;
```

```
if (rootCount > 0) { docsInStatus = 0;}
while(docsInStatus == 0) {

   PSJBIDocs docAccountInfo = docsIn.GetDoc("Account_Info_List");
   int count = docsIn.GetCount("Account_Info_List");
   for(int i = 0; i < count; i++) {
      try
      {
      str1 = docAccountInfo.GetValue("Account_Number");
      str2 = docAccountInfo.GetValue("Meter_Number");
      str3 = docAccountInfo.GetValue("Service_Type");
      docAccountInfo.GetNextDoc();
      }
      catch ( NoObjectReferenceException e)
      {
         return PSReturnStatus.FAILED;
      }
   }

   // Call the external system and put the resulting values into the PSJBIDocs
   // output documents (code not shown) */

   docsInStatus = docsIn.GetNextDoc();
} //end while
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
docAccountInfo.destroy();
docsIn.destroy();
```

The figure below shows the input document for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List. This is a version of the Freight Carrier plug-in that was edited for this example.



Example CBIDocs Input Document

GetDoc

Syntax

C++:

```
CBIDocs GetDoc(const TCHAR* docname);
```

Visual Basic:

```
GetDoc(docname As String) As PsBIDocs
```

Java:

```
PSJBIDocs GetDoc(String docname);
```

Description

Gets a document from an input document. The document is an input parameter for a Business Interlink object that is not of simple type (such as integer or string). You must get the document from the input document before you can get values from its members with **GetValue**.

You can call **GetDoc** using a nesting feature. This feature allows you to access deeply nested documents with one call to **GetDoc**, rather than calling **GetDoc** for each nesting. See the Example section below for an example of this.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

Parameter	Description
docname	The name of the input document that **GetDoc** gets.

Returns

The document received from the input document.

Value	Meaning
CBIDocs, PsBIDocs, PSJBIDocs	The document received from the input document.

Example

In the following example, the input document for Calculate Cost(Domestic), or the root level document, is created with the **GetInputDocs** method. The Calculate Cost input parameter From is a document, so the **GetDoc** method gets it from the input document.

C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();
```

```
while(docsIn.GetStatus() == eNoError) {

   PSIOString str0 = docsIn.GetValue(_T("input_param1"));

   CBIDocs docFrom = docsIn.GetDoc(_T("From"));
   PSIOString strPCode = docFrom.GetValue(_T("OriginPostal_Code"));
   PSIOString strCountry = docFrom.GetValue(_T("OriginCountry"));

/* Call GetDoc, GetValue for To, Package_Info, call GetDoc, GetValue,?
 GetNextDoc for Account_Info_List (code not shown) */

/* Call the external system and put the resulting values into the?
 CBIDocs output documents (code not shown) */

   docsIn.GetNextDoc();
} //end while
```

Visual Basic:

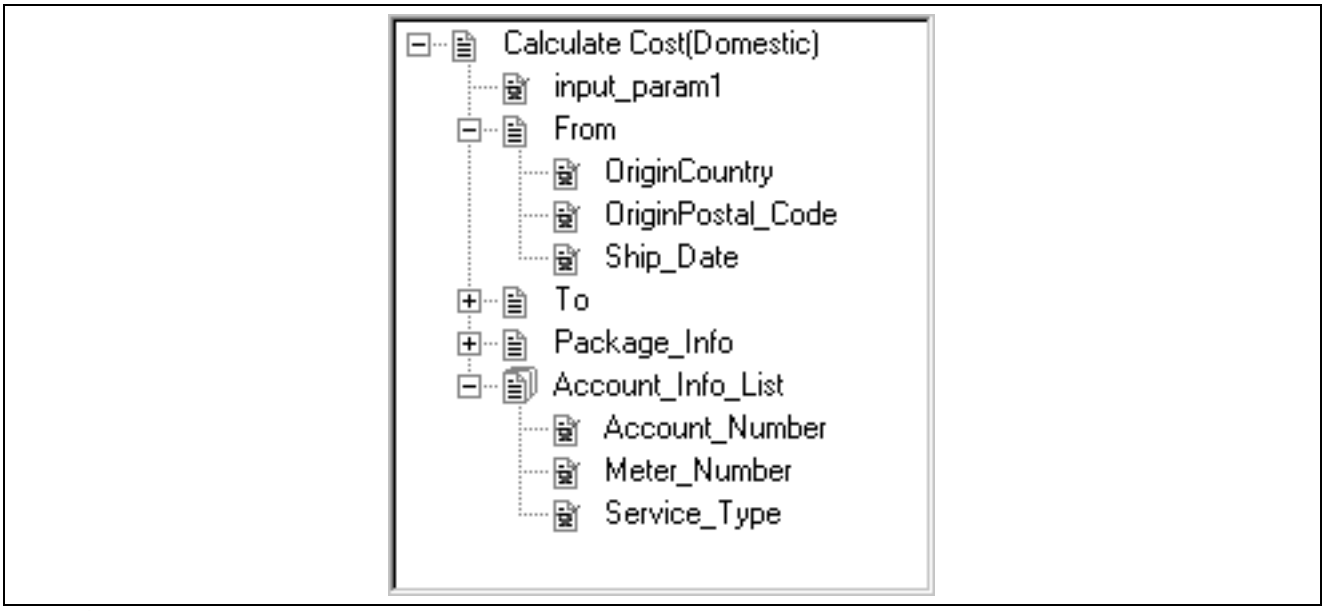
```
Dim str0 As String
Dim str1 As String
Dim str2 As String
Dim objInputDocs As PsBIDocs
Dim objDocFrom As PsBIDocs

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor

While objInputDocs.GetStatus = eNoError

   str0 = objInputDocs.GetValue("input_param1")

   Set objDocFrom = objInputDocs.GetDoc("From")
   str1 = objDocFrom.GetValue("OriginPostal_Code")
   str2 = objDocFrom.GetValue("OriginCountry")

' Call GetDoc, GetValue for To, Package_Info, call GetDoc, GetValue,
' GetNextDoc for Account_Info_List (code not shown)

' Call the external system and put the resulting values into the
' CBIDocs output documents (code not shown)

   objInputDocs.GetNextDoc
Wend
```

Java:

```
PSJBIDocs docsIn = intobj.GetInputDocs("");
docsIn.ResetCursor();
String str0 = new String();
String strPCode = new String();
String strCountry = new String();
```

```
// GetCount method, docsInStatus variable used instead of GetStatus
int rootCount = docsIn.GetCount("root");
int docsInStatus = -1;
if (rootCount > 0) { docsInStatus = 0;}
while(docsInStatus == 0) {

   str0 = docsIn.GetValue("input_param1");

   PSJBIDocs docFrom = docsIn.GetDoc("From");
   strPCode = docFrom.GetValue("OriginPostal_Code");
   strCountry = docFrom.GetValue("OriginCountry");

/* Call GetDoc, GetValue for To, Package_Info, call GetDoc, GetValue,?
 GetNextDoc for Account_Info_List (code not shown) */

/* Call the external system and put the resulting values into the?
 CBIDocs output documents (code not shown) */

   docsInStatus = docsIn.GetNextDoc();
} //end while
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
docFrom.destroy();
docsIn.destroy();
```

The figure below shows the CBIDocs input document for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List. This is a version of the Freight Carrier plug-in that was edited for this example.



Example CBIDocs Input Document

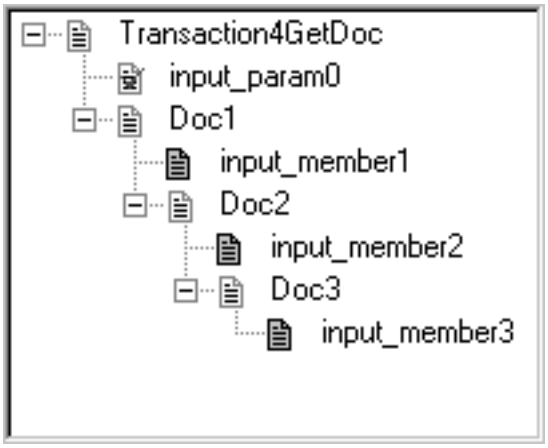
In the following C++ examples, **GetDoc** is used to access a document that is nested more deeply. If you want to access a document that is more deeply nested, you can either call **GetDoc** for each nesting, or you can call **GetDoc** once using the nesting feature.

Calling **GetDoc** with the nesting feature:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

CBIDocs Docs3 = docsIn.GetDoc(_T("Doc1.Doc2.Doc3"));
PSIOString strIn3 = Docs3.GetValue(_T("input_member3"));
```

Calling **GetDoc** without the nesting feature:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

CBIDocs Docs1 = docsIn.GetDoc(_T("Doc1"));
CBIDocs Docs2 = Docs1.GetDoc(_T("Doc2"));
CBIDocs Docs3 = Docs2.GetDoc(_T("Doc3"));
PSIOString strIn3 = Docs3.GetValue(_T("input_member3"));
```



Example BIDocs Input Document: Nested Documents

GetNextDoc

Syntax

C++:

```
int GetNextDoc()
```

Visual Basic:

```
GetNextDoc() As Long
```

Java:

```
int GetNextDoc()
```

Description

The **GetNextDoc** method gets the next document from one of the following levels:

- The root level of the input document for a Business Interlink object. This level was created with the method **GetInputDocs**.
- When a document within the input document is a list, **GetNextDoc** gets another document from the list. If you use **GetNextDoc** on a document that is not a list, **GetNextDoc** fails and returns an error.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

None.

Returns

An integer with the following possible values:

Number	Enum value and Meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.

Number	Enum value and Meaning
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

In this example, the input document for Calculate Cost, or the root level document, is created with the **GetInputDocs** method. The **GetNextDoc** method gets another root level document. The Calculate Cost output parameter Account_Info_List is a document, so the **GetDoc** method gets it from the input document. Account_Info_List is also a document list, so **GetNextDoc** method gets the next Account_Info_List document.

C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

while(docsIn.GetStatus() == eNoError) {

   const TCHAR *str0 = docsIn.GetValue(_T("input_param1"));

   /* Call GetDoc, GetValue for From, To, Package_Info (code not
      shown) */

   CBIDocs docAccountInfo = docsIn.GetDoc(_T("Account_Info_List"));
   int count = docsIn.GetCount(_T("Account_Info_List"));
   for(int i = 0; i < count; i++) {
      const TCHAR *str1 =
         docAccountInfo.GetValue(_T("Account_Number"));
      const TCHAR *str2 =
         docAccountInfo.GetValue(_T("Meter_Number"));
      const TCHAR *str3 =
         docAccountInfo.GetValue(_T("Service_Type"));
      docAccountInfo.GetNextDoc();
   }

/* Call the external system and put the resulting values into the CBIDocs output?
 documents (code not shown) */

   docsIn.GetNextDoc();
} //end while
```

Visual Basic:

```
Dim str1 As String
Dim str2 As String
Dim str3 As String
Dim count As Long
```

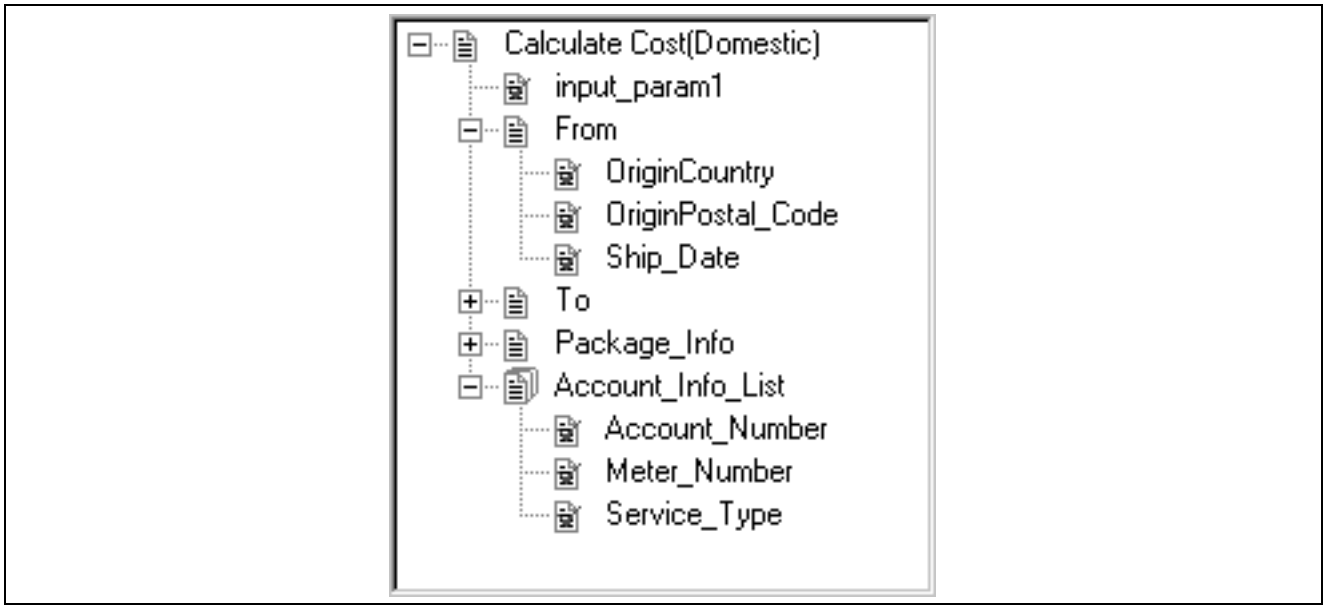
```
Dim lIndex As Long
Dim objInputDocs As PsBIDocs
Dim objDocAccountInfo As PsBIDocs

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor

' Call GetValue for input_param1, GetDoc, GetValue for From, To,
' Package_Info (code not shown) */

While objInputDocs.GetStatus = eNoError

   Set objDocAccountInfo = objInputDocs.GetDoc("Account_Info_List")
   count = objInputDocs.GetCount("Account_Info_List")
   For lIndex = 1 To count
      str1 = objDocAccountInfo.GetValue("Account_Number")
      str2 = objDocAccountInfo.GetValue("Meter_Number")
      str3 = objDocAccountInfo.GetValue("Service_Type")
      objDocAccountInfo.GetNextDoc
   Next

' Call the external system and put the resulting values into the
' output documents (code not shown)

   objInputDocs.GetNextDoc
Wend
```

Java:

```
PSJBIDocs docsIn = intobj.GetInputDocs("");
docsIn.ResetCursor();
String str0 = new String();
String str1 = new String();
String str2 = new String();
String str3 = new String();

// GetCount method, docsInStatus variable used instead of GetStatus
int rootCount = docsIn.GetCount("root");
int docsInStatus = -1;
if (rootCount > 0) { docsInStatus = 0;}
while(docsInStatus == 0) {

   str0 = docsIn.GetValue("input_param1");

   // Call GetDoc, GetValue for From, To, Package_Info (code not
   // shown)

   PSJBIDocs docAccountInfo = docsIn.GetDoc("Account_Info_List");
   int count = docsIn.GetCount("Account_Info_List");
   for(int i = 0; i < count; i++) {
      try
```

```
        {
        str1 = docAccountInfo.GetValue("Account_Number");
        str2 = docAccountInfo.GetValue("Meter_Number");
        str3 = docAccountInfo.GetValue("Service_Type");
        docAccountInfo.GetNextDoc();
        }
        catch ( NoObjectReferenceException e)
        {
           return PSReturnStatus.FAILED;
        }
     }

     // Call the external system and put the resulting values into the PSJBIDocs
     // output documents (code not shown) */

     docsInStatus = docsIn.GetNextDoc();
  } //end while
  // Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
  docAccountInfo.destroy();
  docsIn.destroy();
```

The figure below shows the input document for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List. This is a version of the Freight Carrier plug-in that was edited for this example.



Example CBIDocs Input Document

GetPreviousDoc

Syntax

C++:

```
int GetPreviousDoc();
```

Visual Basic:
```
GetPreviousDoc() As Long
```

Java:
```
int GetPreviousDoc();
```

Description

The **GetPreviousDoc** method gets the previous document from one of the following levels:

- The root level of the input document for a Business Interlink object. This level was created with the method **GetInputDocs**.
- When a document within the input document is a list, **GetPreviousDoc** gets another document from the list. If you use **GetPreviousDoc** on a document that is not a list, **GetPreviousDoc** fails and returns an error.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

None.

Returns

An integer with the following possible values:

Number	Enum value and Meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.

Number	Enum value and Meaning
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

In the following example, the input document for Calculate Cost, or the root level document, is created with the **GetInputDocs** method. The Calculate Cost input parameter Account_Info_List is a document list, so the **GetDoc** method gets it. The **GetCount** and **MoveToDoc** methods point to the last Account_Info_List document in the list. The **GetPreviousDoc** method is used in a loop to cycle through the Account_Info_List list, starting with the last and ending with the first in the list, and get each Account_Info_List document and its values.

C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

while(docsIn.GetStatus() == eNoError) {

   CBIDocs docAccountInfo = docsIn.GetDoc(_T("Account_Info_List"));
   int count = docsIn.GetCount(_T("Account_Info_List"));
   docAccountInfo.MoveToDoc(count-1);
   for(int i = count; i > 0; i--) {
      const TCHAR *str1 =
         docAccountInfo.GetValue(_T("Account_Number"));
      const TCHAR *str2 =
         docAccountInfo.GetValue(_T("Meter_Number"));
      const TCHAR *str3 =
         docAccountInfo.GetValue(_T("Service_Type"));
      docAccountInfo.GetPreviousDoc();
   }

/* Call the external system and put the resulting values into the CBIDocs output?
```

```
 documents (code not shown) */

   docsIn.GetNextDoc();
} //end while
```

Visual Basic:

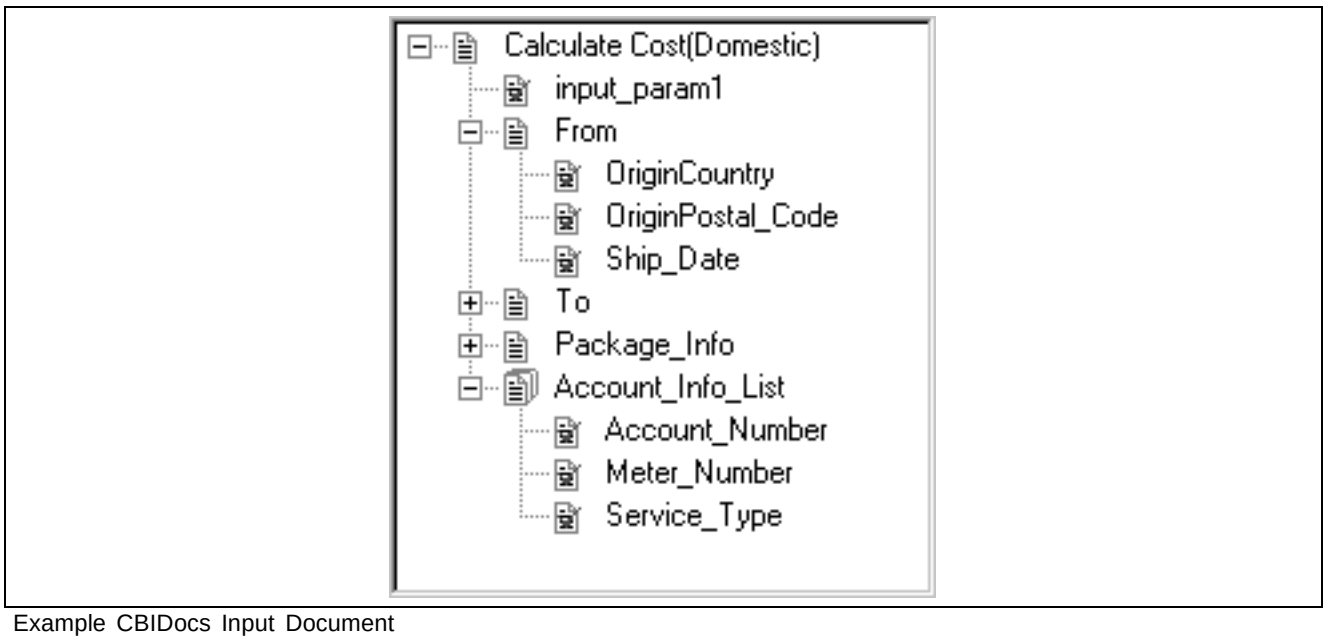
```
Dim str1 As String
Dim str2 As String
Dim str3 As String
Dim count As Long
Dim lIndex As Long
Dim objInputDocs As PsBIDocs
Dim objDocAccountInfo As PsBIDocs

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor

' Call GetValue for input_param1, GetDoc, GetValue for From, To,
' Package_Info (code not shown) */

While objInputDocs.GetStatus = eNoError

   Set objDocAccountInfo = objInputDocs.GetDoc("Account_Info_List")
   count = objInputDocs.GetCount("Account_Info_List")
   objDocAccountInfo.MoveToDoc(count-1)
   For lIndex = count To 1 Step -1
      str1 = objDocAccountInfo.GetValue("Account_Number")
      str2 = objDocAccountInfo.GetValue("Meter_Number")
      str3 = objDocAccountInfo.GetValue("Service_Type")
      objDocAccountInfo.GetPreviousDoc
   Next

' Call the external system and put the resulting values into the
' output documents (code not shown)

   objInputDocs.GetNextDoc
Wend
```

Java:

```
PSJBIDocs docsIn = intobj.GetInputDocs("");
docsIn.ResetCursor();
String str1 = new String();
String str2 = new String();
String str3 = new String();

// GetCount method, docsInStatus variable used instead of GetStatus
int rootCount = docsIn.GetCount("root");
int docsInStatus = -1;
if (rootCount > 0) { docsInStatus = 0;}
while(docsInStatus == 0) {
```

```
   PSJBIDocs docAccountInfo = docsIn.GetDoc("Account_Info_List");
   int count = docsIn.GetCount("Account_Info_List");
   docAccountInfo.MoveToDoc(count-1);
   for(int i = count; i > 0; i--) {
      try
      {
      str1 = docAccountInfo.GetValue("Account_Number");
      str2 = docAccountInfo.GetValue("Meter_Number");
      str3 = docAccountInfo.GetValue("Service_Type");
      docAccountInfo.GetPreviousDoc();
      }
      catch ( NoObjectReferenceException e)
      {
         return PSReturnStatus.FAILED;
      }
   }

   // Call the external system and put the resulting values into the PSJBIDocs
   // output documents (code not shown) */

   docsInStatus = docsIn.GetNextDoc();
} //end while
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
docAccountInfo.destroy();
docsIn.destroy();
```

The figure below shows the CBIDocs input document for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List. This is a version of the Freight Carrier plug-in that was edited for this example.



Example CBIDocs Input Document

GetStatus

Syntax

C++:

```
int GetStatus()
```

Visual Basic:

```
GetStatus() As Long
```

Description

Tests the document for a Business Interlink object. Returns the status of the document.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs

Parameters

None.

Returns

An integer with the following possible values:

Number	Enum value and Meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.

Number	Enum value and Meaning
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

The following example uses the **GetStatus** method to test docsIn, which is a CBIDocs input document for a Business Interlink object, to see if it contains data. If it does, the **GetNextDoc** method gets a CBIDocs input document, allowing you to later get its values.

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

while(docsIn.GetStatus() == eNoError)
{
   // Get input values and process them
   // Insert output values
   docsIn.GetNextDoc();
}
```

Visual Basic:

```
Dim eNoError As Long
Dim objInputDocs As PsBIDocs

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor

While objInputDocs.GetStatus = eNoError

   ' Get input values and process them
   ' Insert output values
   objInputDocs.GetNextDoc
Wend
```

GetValue

Syntax

C++:

```
const TCHAR* GetValue(const TCHAR* name);
const TCHAR* GetValue(const PSIOString & name){return GetValue(strName.c_str());}
const int GetValue(const TCHAR* name, int&);
const int GetValue(const TCHAR* name, float&);
const int GetValue(const TCHAR* name, double&);
```

Visual Basic:

```
GetValue(name As String) As String
GetValueDouble(name As String, value As Double) As Long
GetValueFloat(name As String, value As Single) As Long
GetValueInt(name As String, value As Long) As Long
```

Java:

```
String GetValue(String name);
int GetValue(String name, Integer value);
int GetValue(String name, Float value);
int GetValue(String name, Double value);
int GetValue(String name, String time);
```

Description

Gets a value from an input parameter within a document of the input document for a Business Interlink object.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

Parameter	Description
name	The name of the member of the document that is having its value retrieved.
value, int&, float&, double&, time	The retrieved value.

Returns

The value of the input parameter.

Value	Meaning
value, int&, float&, double&	The value of the input parameter.
int	When the syntax is int GetValue(*name*, *value*), int is the return status of GetValue, listed in the table below.

An int can return the following values:

Number	Enum value and Meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

In the following C++ example, the input document for Calculate Cost(Domestic), or the root level document, is created with the **GetInputDocs** method. The GetValue method gets the value of the input_param1 input parameter. The Calculate Cost input parameter From is a document, so the **GetDoc** method gets it from the input document. Then **GetValue** gets the values of the From members.

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

while(docsIn.GetStatus() == eNoError) {

   PSIOString str0 = docsIn.GetValue(_T("input_param1"));

   CBIDocs docFrom = docsIn.GetDoc(_T("From"));
   PSIOString strPCode = docFrom.GetValue(_T("OriginPostal_Code"));
   PSIOString strCountry = docFrom.GetValue(_T("OriginCountry"));

/* Call GetDoc, GetValue for To, Package_Info, call GetDoc, GetValue, GetNextDoc?
 for Account_Info_List (code not shown) */

/* Call the external system and put the resulting values into the CBIDocs output?
 documents (code not shown) */

   docsIn.GetNextDoc();
} //end while
// Use one destroy for each GetInputDocs&GetNextDoc
docsIn.destroy();
```

In the following Visual Basic example, the input document for Calculate Cost(Domestic), or the root level document, is created with the **GetInputDocs** method. The GetValue method gets the value of the input_param1 input parameter. The Calculate Cost input parameter From is a document, so the **GetDoc** method gets it from the input document. Then **GetValue** gets the values of the From members.

```
Dim str0 As String
Dim strPCode As String
Dim strCountry As String
Dim objInputDocs As PsBIDocs
Dim objDocFrom As PsBIDocs

Set objInputDocs = pIntObj.GetInputDocs
objInputDocs.ResetCursor

While objInputDocs.GetStatus = eNoError

   str0 = objInputDocs.GetValue("input_param1")

   Set objDocFrom = objInputDocs.GetDoc("From")
   strPCode = objDocFrom.GetValue("OriginPostal_Code")
   strCountry = objDocFrom.GetValue("OriginCountry")

' Call GetDoc, GetValue for To, Package_Info, call GetDoc, GetValue,
```

```
' GetNextDoc for Account_Info_List (code not shown) */

' Call the external system and put the resulting values into the
' output documents (code not shown) */

   objInputDocs.GetNextDoc
Wend
```

In the following Java example, the input document for Calculate Cost(Domestic), or the root level document, is created with the **GetInputDocs** method. The GetValue method gets the value of the input_param1 input parameter. The Calculate Cost input parameter From is a document, so the **GetDoc** method gets it from the input document. Then **GetValue** gets the values of the From members.

```
try {

   PSJBIDocs docsIn = intobj.GetInputDocs("");
   docsIn.ResetCursor();

   // GetCount method, docsInStatus variable used instead of GetStatus
   int rootCount = docsIn.GetCount("root");
   int docsInStatus = -1;
   if (rootCount > 0) { docsInStatus = 0;}
   while(docsInStatus == 0) {

   inputstring = docsIn.GetValue("input_param1");

   PSJBIDocs docFrom = docsIn.GetDoc("From");
   String strPCode = docFrom.GetValue("OriginPostal_Code");
   String strCountry = docFrom.GetValue("OriginCountry");

/* Call GetDoc, GetValue for To, Package_Info, call GetDoc, GetValue, GetNextDoc?
 for Account_Info_List (code not shown) */

/* Call the external system and put the resulting values into the CBIDocs output?
 documents (code not shown) */

   docsInStatus = docsIn.GetNextDoc();
} //end while
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
docFrom.destroy();
docsIn.destroy();
```

MoveToDoc

Syntax

C++:

```
int MoveToDoc(int iIndex);
```

Visual Basic:

```
MoveToDoc(iIndex As Long) As Long
```

Java:

```
int MoveToDoc(int iIndex);
```

Description

Within a list of documents in the output document, the **MoveToDoc** method moves to the documents given by the parameter *list_number*. After using **MoveToDoc**, the **GetValue** method gets the values of the document that is in the *list_number*+1 location in the list.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

Parameter	Description
iIndex	The number indicating the document that MoveToDoc moves to. After using MoveToDoc, the GetValue method gets the values of the document that is in the list_number+1 location in the list. For example, if list_number is zero, then MoveToDoc moves to the first document in the list.

Returns

An integer with the following possible values:

Number	Enum value and Meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
2	LIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time will result in this error.
3	DOCUMENT_UNINITIALIZED. Internal error.
4	NOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.

Number	Enum value and Meaning
6	NOT_LISTTYPE. You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE. You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

In the following example, the input document for Calculate Cost, or the root level document, is created with the **GetInputDocs** method. The Calculate Cost input parameter Account_Info_List is a document list, so the **GetDoc** method gets it. The **GetCount** and **MoveToDoc** methods point to the last Account_Info_List document in the list. The **GetValue** method then gets the values for the last document in the list.

C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();

CBIDocs docAccountInfo = docsIn.GetDoc(_T("Account_Info_List"));
int count = docsIn.GetCount(_T("Account_Info_List"));
docAccountInfo.MoveToDoc(count-1);
const TCHAR *str1 = docAccountInfo.GetValue(_T("Account_Number"));
const TCHAR *str2 = docAccountInfo.GetValue(_T("Meter_Number"));
const TCHAR *str3 = docAccountInfo.GetValue(_T("Service_Type"));
```

Visual Basic:

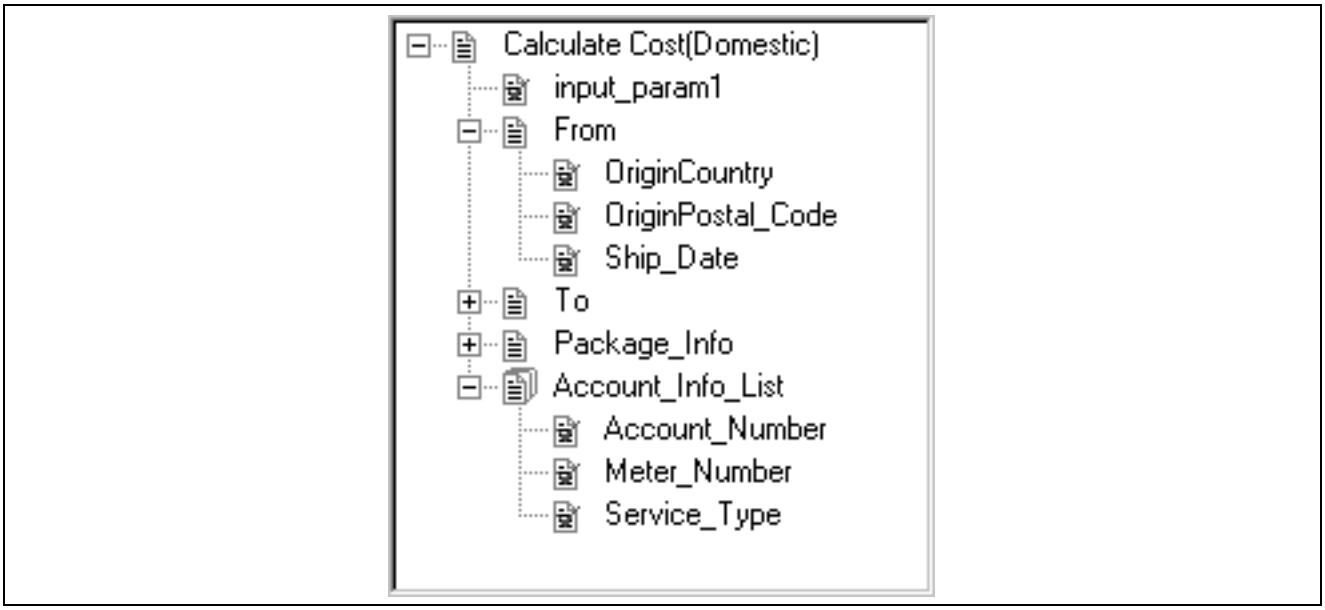
```
Dim str1 As String
Dim str2 As String
Dim str3 As String
Dim objInputDocs As PsBIDocs
Dim objDocAccountInfo As PsBIDocs

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor


Set objDocAccountInfo = objInputDocs.GetDoc("Account_Info_List")
   count = objInputDocs.GetCount("Account_Info_List")
objDocAccountInfo.MoveToDoc(count-1)
str1 = objDocAccountInfo.GetValue("Account_Number")
```

```
str2 = objDocAccountInfo.GetValue("Meter_Number")
str3 = objDocAccountInfo.GetValue("Service_Type")
```

Java:

```
PSJBIDocs docsIn = intobj.GetInputDocs("");
docsIn.ResetCursor();

PSJBIDocs docAccountInfo = docsIn.GetDoc("Account_Info_List");
int count = docsIn.GetCount("Account_Info_List");
docAccountInfo.MoveToDoc(count-1);
String str1 = docAccountInfo.GetValue("Account_Number");
String str2 = docAccountInfo.GetValue("Meter_Number");
String str3 = docAccountInfo.GetValue("Service_Type");
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
docAccountInfo.destroy();
docsIn.destroy();
```

The figure below shows the input document for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List. This is a version of the Freight Carrier plug-in that was edited for this example.



Example CBIDocs Input Document

ResetCursor

Syntax

C++:

```
void ResetCursor();
```

Visual Basic:

```
ResetCursor()
```

Description

Resets the cursor in the CBIDocs input document for a Business Interlink object to the top. Once you call this method, the next time you call **GetValue**, you get an input value from the first document of the CBIDocs input documents for a Business Interlink object.

The classes are:

- C++: CBIDocs
- Visual Basic: PsBIDocs
- Java: PSJBIDocs

Parameters

None.

Returns

None.

Example

The following code uses **ResetCursor** to reset the cursor in the input document to the top. IntObj is a pointer to a Business Interlink object.

C++:

```
CBIDocs docsIn = IntObj->GetInputDocs(_T(""));
docsIn.ResetCursor();
```

Visual Basic:

```
Dim objInputDocs As PsBIDocs

Set objInputDocs = IntObj.GetInputDocs
objInputDocs.ResetCursor
```

Java:

```
PSJBIDocs docsIn = intobj.GetInputDocs("");
docsIn.ResetCursor();

// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc
docsIn.destroy();
```

Accessing Input/Output Parameter Information With PSIOString methods (C++ only)

This section discusses the PSIOString methods, and lists and discusses each method.

PSIOString Methods

You use the PSIOString methods to access the information in a PSIOString, which is how, in C++, the input and output parameter information for a Business Interlink object is stored.

You must include the header file ioutil.h for this class.

The following list shows which PSIOString methods you use to operate on the input and output values.

- Use append or the operators += or + to append a string.
- Use c_str to get a TCHAR string from a string.
- Use find to find a character in a string.
- Use length to find the length of a string in characters.
- Use rfind to find a character in a string, starting from the right end of the string.
- Use size to get the number of characters in a string or the number of input and output parameters.
- Use substr to get a substring from a string.
- Use the operators == or != to perform comparisons on strings.

append

Syntax

C++:

Class: PSIOString

```
CPSIOString& append(PSIOString);
CPSIOString& append(TCHAR);
```

Description

Appends a string to the end of the PSIOString string.

Parameters

Parameter	Description
TCHAR	The string that you want to append to this string.

Returns

The appended string.

Value	Meaning
CPSIOString	The appended string.

Example

The following code uses the append method to append a comma to the end of the PSIOString named string.

```
string = string.append(_T(","));
```

c_astr

Syntax

C++:

```
char& c_astr();
```

Description

Converts a PSIOString string to a char string.

The data for a Business Interlink object is stored entirely in Unicode. If your external system uses char data rather than TCHAR data, its functions will require char data. Use the c_astr method (instead of the c_str method) to pass the Business Interlink data to your external system functions.

The class is PSIOString.

Parameters

None.

Returns

The char string.

Value	Meaning
char	The char string, converted from the PSIOString.

Example

The following code shows the c_astr method being used to extract input parameters values as char values. This example is an edited version of the PSCustomer example.

```
while(inBuf.GetNextRow(inRow))
{
   char *strFileName = "c:\\temp\\abc.tmp";

   char *strPostData = "origin=";
   strcpy(strPostData, inRow[0].c_astr());
   strcpy(strPostData, "&dest=");
   strcpy(strPostData, inRow[1].c_astr());

// Perform calculations based upon the inputs
}
```

c_str

Syntax

C++:

```
TCHAR& c_str();
```

Description

Converts a PSIOString string to a TCHAR string. Use this method to extract data from the Business Interlink object input/output table when the input/output for your Business Interlink plug-in is stored as Unicode.

The data for a Business Interlink object is stored entirely in Unicode. If your software system uses TCHAR data, use the c_astr method (instead of the c_astr method) to pass the Business Interlink data to your software system functions.

The class is PSIOString.

Parameters

None.

Returns

The TCHAR string.

Value	Meaning
TCHAR	The TCHAR string, converted from the PSIOString.

Example

The following code uses the c_str method to convert the PSIOString named varListOutput[i]->m_strName so that it can be used as an input for the method AddColumn. A VARINFORLIST consists of rows, and each row contains information about one input or output parameter, including PSIOString m_strName: the name of the parameter.

```
const VARINFOLIST& varListOutput = m_pIntObj->GetOutputParams();
for(int i = 0; i < varListOutput.size(); i++)
{
   outBuf.AddColumn(varListOutput[i]->m_strName.c_str());
}
```

find

Syntax

C++:

```
int find(PSIOString);
int find(TCHAR);
```

Description

Finds the location of a string within this PSIOString or TCHAR, searching from the left end of the PSIOString or TCHAR.

The class is PSIOString.

Parameters

Parameter	Description
PSIOString or TCHAR	The string to search for.

Returns

The location within the PSIOString of the searched-for string.

Value	Meaning
int	The location within the PSIOString of the searched-for string. The default value is 0.

Example

The following code uses the find method to search for a period in each output parameter name, starting from the left.

```
const VARINFOLIST& varListOutput = m_pIntObj->GetOutputParams();
for(int i = 0; i < varListOutput.size(); i++)
{
   int index = varListOutput[i]->m_strName.rfind(_T('.'));
   if (index != 0)
   {
     /* Perform a process here if you found a period in the name. */
   }
}
```

length

Syntax

C++:

```
int length();
```

Description

Finds the length of this PSIOString in characters.

The class is PSIOString.

Parameters

None.

Returns

The length of this PSIOString in characters.

Value	Meaning
int	The length of this PSIOString in characters.

Example

The following code uses the find method to search for a period in each output parameter name, starting from the left.

```
const VARINFOLIST& varListOutput = m_pIntObj->GetOutputParams();
for(int i = 0; i < varListOutput.size(); i++)
{
   int index = varListOutput[i]->m_strName.rfind(_T('.'));
   if (index != 0)
   {
     /* Perform a process here if you found a period in the name. */
   }
}
```

rfind

Syntax

C++:

```
int rfind(PSIOString);
int rfind(TCHAR);
```

Description

Finds the location of a string within this PSIOString, searching from the right end of the PSIOString.

The class is PSIOString.

Parameters

Parameter	Description
PSIOString or TCHAR	The string to search for.

Returns

The location within the PSIOString of the searched-for string.

Value	Meaning
int	The location within the PSIOString of the searched-for string. The default value is -1.

Example

If an output parameter is a class (m_DataType = IOC_DATATYPE_OBJECT), you will retrieve the names of the class member. The name will be of the form *class.member*, where *class* is the name of the class, and *member* is the name of the class member. In that case, if you want to use only the name of the member without the class (*member*, not *class.member*), you can use the CPSIOBuf methods rfind and substr to extract it.

The following code checks the output parameter names for the *class.member* format. It still stores the parameter name in the *class.member* format. However, if it finds a period in the parameter name, it strips off *class.* from the output parameter name, leaving only *member*. *member* is then used in the comparison to set the indexes.

```
const VARINFOLIST& varListOutput = m_pIntObj->GetOutputParams();
for(int i = 0; i < varListOutput.size(); i++)
{
   outBuf.AddColumn(varListOutput[i]->m_strName.c_str());
   PSIOString strTemp;
   int index = varListOutput[i]->m_strName.rfind(_T('.'));

   if(index == -1)
      strTemp = varListOutput[i]->m_strName;
   else
      strTemp = varListOutput[i]->m_strName.substr(index+1);

   if(strTemp ==      _T("sender"))            iSender = i;
   else if(strTemp == _T("date"))              iDate = i;
   else if(strTemp == _T("subject"))           iSubject = i;
   else if(strTemp == _T("message_id"))        iId = i;
   else if(strTemp == _T("body"))              iBody = i;
   else if(strTemp == _T("return_status_msg")) iReturnMessage = i;
   else if(strTemp == _T("return_status"))     iReturnStatus = i;
}
```

size

Syntax

C++:

```
int size();
```

Description

For class PSIOString, which contains a Unicode string, returns the number of characters in the string.

For class VARINFOLIST, which contains the input/output parameter information, returns the number of input/output parameters.

For class IOSTRINGLIST, which contains a row of input/output parameter values, returns the number of input/output parameters.

Parameters

None.

Returns

The number of characters or parameters.

Value	Meaning
int	For PSIOString, which contains a Unicode string, returns the number of characters in the string. For VARINFORLIST, which contains the input/output parameter information, returns the number of input/output parameters. For IOSTRINGLIST, which contains a row of input/output parameter values, returns the number of input/output parameters.

Example

The following code uses the size method to set the number of output parameters in a for loop.

```
const VARINFOLIST& varListOutput = m_pIntObj->GetOutputParams();
for(int i = 0; i < varListOutput.size(); i++)
{
   int index = varListOutput[i]->m_strName.rfind(_T('.'));
   if (index != 0)
   {
     /* Perform a process here if you found a period in the name. */
   }
}
```

substr

Syntax

C++:

```
PSIOString substr(int);
```

Description

Returns a string containing the contents of this string, from the location given to the end of the string.

The class is PSIOString.

Parameters

Parameter	Description
Int	The location in the PSIOString from which to extract the remainder of the string.

Returns

A string containing the contents of this string, from the location given to the end of the string.

Value	Meaning
PSIOString	A string containing the contents of this string, from the location given to the end of the string. For example, if the PSIOString name contained *class.member*, then name.substr[7] would contain *member*.

Example

If an output parameter is a class (m_DataType = IOC_DATATYPE_OBJECT), you will retrieve the names of the class member. The name will be of the form *class.member*, where *class* is the name of the class, and *member* is the name of the class member. In that case, if you want to use only the name of the member without the class (*member*, not *class.member*), you can use the CPSIOBuf methods rfind and substr to extract it.

The following code checks the output parameter names for the *class.member* format. It still stores the parameter name in the *class.member* format. However, if it finds a period in the parameter name, it strips off *class.* from the output parameter name, leaving only *member*. *member* is then used in the comparison to set the indexes.

```
const VARINFOLIST& varListOutput = m_pIntObj->GetOutputParams();
for(int i = 0; i < varListOutput.size(); i++)
{
   outBuf.AddColumn(varListOutput[i]->m_strName.c_str());
   PSIOString strTemp;
   int index = varListOutput[i]->m_strName.rfind(_T('.'));

   if(index == -1)
      strTemp = varListOutput[i]->m_strName;
   else
      strTemp = varListOutput[i]->m_strName.substr(index+1);

   if(strTemp ==      _T("sender"))            iSender = i;
   else if(strTemp == _T("date"))              iDate = i;
   else if(strTemp == _T("subject"))           iSubject = i;
   else if(strTemp == _T("message_id"))        iId = i;
   else if(strTemp == _T("body"))              iBody = i;
   else if(strTemp == _T("return_status_msg")) iReturnMessage = i;
   else if(strTemp == _T("return_status"))     iReturnStatus = i;
}
```

operators +, +=, ==, !=

Description

Class: PSIOString

These operators allow the following operations to be performed upon PSIOString and TCHAR.

Operator	Description
++	Append one PSIOString to another PSIOString.
+	Append two PSIOStrings and store the result in a third PSIOString.
==	equal comparison
!=	not equal comparison

Using Parameter Lists

The configuration parameters, input parameters, and output parameters from the Interlink Object all use a data structure list to store parameter information. C++ uses VARINFOLIST, and Visual Basic uses IPsEnumVarInfo. Each item in the IPsEnumVarInfo list is of type VarInfo. Java uses VarInfo to contain one parameter. Each item in the list is one parameter, and has the following structure:

C++: VARINFOLIST	Visual Basic: IPsEnumVarInfo	Java: VarInfo	Meaning
PSIOString m_strName	string Name	String strName()	the name of the parameter.
PSIOString m_strDefault	string Value	String strDefault()	any default value for the parameter.
DataType m_DataType	ENUM_ EIOCDATATYPE DataType	int DataType	the data type of the parameter.
EVARATTRIBUTE m_nAttribute	ENUM_ VARATTRIBUTE Attribute	int nAttribute()	indicates if the parameter is required.

C++: VARINFOLIST	Visual Basic: IPsEnumVarInfo	Java: VarInfo	Meaning
PSIOString m_ strPathName	String PathInfo	String strPathInfo()	the entire path of this parameter. For example, if this parameter is the string Country in the document From, where From is a class of type Origin and Country is an enum with values of United States and Puerto Rico, the path is object<Origin>.enum< United States,Puerto Rico>. If this parameter is the string Postal_Code in the document From, where From is a class of type Origin and Postal_Code is a string, the path is object<Origin>.string.
PSIOString m_ strDisplayName	String DisplayName	String strDisplayName	This name shows in the Input Name and Output Name columns in the Application Designer for Business Interlinks.

The valid values for m_DataType or DataType are:

Data Type: C++, Visual Basic, Java	Definition and format
IOC_DATATYPE_INT, ENUM_IOC_DATATYPE_INT	An integer.
IOC_DATATYPE_STRING, ENUM_IOC_ DATATYPE_STRING	A character string.
IOC_DATATYPE_FLOAT, ENUM_IOC_DATATYPE_ FLOAT	A floating point variable.
IOC_DATATYPE_BOOLEAN, ENUM_IOC_ DATATYPE_BOOLEAN	A Boolean value of TRUE or FALSE.
IOC_DATATYPE_DATE, ENUM_IOC_DATATYPE_ DATE	A date. The format is YYYY-MM-DD, where YYYY is the year, MM is the month, and DD is the day.
IOC_DATATYPE_TIME, ENUMIOC_DATATYPE_ TIME	A time. The format is HH:MM:SS, where HH is the hour, MM is the minutes, and SS is the seconds.

Data Type: C++, Visual Basic, Java	Definition and format
IOC_DATATYPE_DATETIME, ENUM_IOC_ DATATYPE_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.
IOC_DATATYPE_ENUM, ENUM_IOC_DATATYPE_ ENUM	An enumeration.
IOC_DATATYPE_OBJECT, ENUM_IOC_ DATATYPE_OBJECT	A document. This is either a **<class>** in the XML design-time plug-in, or a class defined by the GetClassByName method. Documents can contain other documents and/or any of the other data types. You use the GetDoc and AddDoc methods to access the contents of documents.
IOC_DATATYPE_PASSWORD, ENUM_IOC_ DATATYPE_PASSWORD	A password. This is a character string.
Lists of the above types, such as IOC_DATATYPE_ LIST_INT, ENUM_IOC_DATATYPE_LIST_INT	Each of the above types except for password can be a list.

CHAPTER 17

Writing the Design-Time Functionality Using Dynamic Catalog Methods

This chapter provides an overview of design-time methods and discusses how to:

- Write the design-time methods.
- Use an example of design-time methods written in Visual Basic.
- Use an example of design-time methods written in C++.
- Use the dynamic catalog methods.

Understanding Design-Time Functionality Using Dynamic Catalog Methods

Once you have designed your transactions and classes by naming the transactions/classes, and naming and typing their parameters, you can 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 either supplying a design-time plug-in, or by writing the design-time functionality into the runtime plug-in. The plug-in is used during the design-time to define and save Business Interlink definitions. It can also be used at runtime to simulate 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 dynamic. A dynamic catalog means that the transactions and classes are not predefined; the input/output parameters for a transaction, or the data members for a class, are changeable in data type or number of parameters/members. (For comparison, static would mean that the transactions and classes are not changeable; a plug-in to a database would not allow the members of a class to be changed, and a new class could not be added or deleted.)

When you use dynamic catalogs, you must implement a set of methods (in C/C++, or any environment supporting COM such as Visual Basic) that are compiled into a library. In this way, your design-time plug-in can connect to the external system and dynamically query its interfaces to provide the most recent additions.

You do not need to write an XML design-time plug-in when you use dynamic catalogs, because the design-time functionality will be contained in the methods described in this chapter. If your catalogs are static, you can write the design-time functionality using either an XML design-time plug-in or the methods described in this chapter.

- You will perform the following actions with these methods:
- Write the categories for the transactions and classes: GetCategories to provide the categories.
- Write the configuration parameters: GetParameterList to provide the names and data types of the configuration parameters.

- Write the class catalog: GetClassByCategory to provide the class names, and GetClassByName to provide the names and data types of the data members of the classes.
- Write the transaction catalog: GetTransactionByCategory to provide the transaction names, and GetTransactionByName to provide the names and data types of the input and output parameters of the transactions.

Writing The Design-Time Methods

This section discusses how to:

- Write the plug-in description using the GetDesc method.
- List the supported Business Interlink types using the IsSupported method.
- List the relational and logical operators (query and update plug-ins only) using the GetCriteriaRelationalOperators and GetCriteriaLogicalOperators methods.
- List the configuration parameters using the GetParameterList method.
- Write the categories for the transactions and classes using the GetCategories method.
- Write the Business Class catalog using the GetClassByCategory and GetClassByName methods.
- Write the Business Transaction catalog using the GetTransactionByCategory and GetTransactionByName methods.
- Write a dynamic catalog.
- Use a Visual Basic example of design methods.
- Use a C++ example of design methods.
- Use the dynamic catalog class methods.

Writing The Plug-in Description: GetDesc

Write the method GetDesc to return a short description of your plug-in. Here is a Visual Basic example.

```
Private Function PsIoDriver_GetDesc() As String
    '
    PsIoDriver_GetDesc = "Freight Carrier"
    '
End Function
```

Here is a C++ example.

```
const TCHAR * SimpleDriver::GetDesc() const
{
return _T("Simple interface driver: adds 2 #, concats 2 strs");
}
```

See Also

Chapter 17, “Writing the Design-Time Functionality Using Dynamic Catalog Methods,” GetDesc, PsIoDriver_GetDesc, page 277

Listing the Supported Business Interlink Types: IsSupported

Write the method IsSupported to return True for every Business Interface Type that your Business Interlink plug-in will support.

Here is an example of IsSupported.

```
Private Function PsIoDriver_IsSupported(ByVal eType As?
 PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED) As Long
    '
    Select Case eType
        Case ENUM_IOC_TRANSACTION
            PsIoDriver_IsSupported = True
        Case ENUM_IOC_STATICCATEGORY
            PsIoDriver_IsSupported = True
        Case ENUM_IOC_ISWCHAR
            PsIoDriver_IsSupported = True
        Case Else
            PsIoDriver_IsSupported = False
    End Select
    '
End Function
```

Here is a C++ example.

```
BOOL SimpleDriver::IsSupported(EIOCINTERFACESUPPORTED option)
{
   switch(option)
   {
      case IOC_TRANSACTION:
      case IOC_STATICCATEGORY:
      return TRUE;

      case IOC_ISWCHAR:
      return  (sizeof(TCHAR) != sizeof(char));

      default:
      return FALSE;
   }
}
```

See Also

Chapter 17, “Writing the Design-Time Functionality Using Dynamic Catalog Methods,” IsSupported, PsIoDriver_IsSupported, page 286

Listing The Configuration Parameters: GetParameterList

Write the method GetParameterList to list 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.

The following example shows the configuration parameter of URL being added to the configuration parameter list.

The configuration parameters are added to the plstParams list, which is a variable of type PSIODRIVERLib.IPsEnumVarInfo. Use the Add method to add the configuration parameters to the plstParams list.

```
Private Function PsIoDriver_GetParameterList(ByVal plstParams As?
 PSIODRIVERLib.IPsEnumVarInfo) As Long
    '
    Dim objVarInfo As VarInfo
    Set objVarInfo = plstParams.Add
    objVarInfo.Name = "URL"
    objVarInfo.DisplayName = ""
    objVarInfo.Value = "file://PSCutomer.dll"
    objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
    objVarInfo.DataDescr = "string"
    objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
    PsIoDriver_GetParameterList = True
    '
End Function
```

The following C++ example shows the configuration parameter of Output_File_Name being added to the configuration parameter list.

The configuration parameters are added to the VARINFOIST list, which contains parameters of type VarInfo. Use the push_back method to add the configuration parameters to the list.

```
VarInfo config(_T("Output_File_Name"), DataType(IOC_DATATYPE_STRING,?
 _T("")), _T(""), _T(""), _T(""), eVAR_ATTR_NONE);

BOOL  SimpleDriver::GetParameterList(VARINFOLIST& list)
{
        list.push_back(&config);
        return TRUE;
}
```

See Also

Chapter 17, “Writing the Design-Time Functionality Using Dynamic Catalog Methods,” GetParameterList, PsIoDriver_GetParameterList, page 281

Writing the Categories For the Transactions and Classes: GetCategories

Write the GetCategories method to return the list of categories for your transactions and/or classes. When you write the IsSupported method to support transactions (ENUM_IOC_TRANSACTION), return a list of your transaction categories; you must at least have the categories “All Transactions” and “Transactions”. When you write the IsSupported method to support classes (ENUM_IOC_OBJQUERY, ENUM_IOC_OBJADD, ENUM_IOC_OBJUPDATE, or ENUM_IOC_OBJDELETE, return a list of your class categories; at least, you must have the category “All Classes” and “Classes”.

The category list is of type plstCriterialNames. Use the plstCriterialNames method Add to add a category name to the list.

The following example creates the two categories All Transactions and Transactions.

```
Private Function PsIoDriver_GetCategories(ByVal eCatType As?
 PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED,?
 ByVal plstCriterialNames As PSIODRIVERLib.IPsEnumString) As Long
    '
    Dim objPsBstr As PsBstr
    Set objPsBstr = plstCriterialNames.Add
    objPsBstr.Data = "Transactions"
    Set objPsBstr = plstCriterialNames.Add
    objPsBstr.Data = "All Transactions"
    PsIoDriver_GetCategories = True
    '
End Function
```

Writing the Business Class Catalog: GetClassByCategory, GetClassByName

Write GetClassByCategory to return a list of classes under the given category name. Since this Business Interlink plug-in only uses classes within transactions, and thus uses no class categories, this method returns False.

```
Private Function PsIoDriver_GetClassByCategory(ByVal bstrFilter As String,?
 ByVal bstrCategory As String,?
 ByVal plstCategory As PSIODRIVERLib.IPsEnumString) As Long
    '
    PsIoDriver_GetClassByCategory = False
    '
End Function
```

Write GetClassByName to return the data members for the given class name.

```
Private Function PsIoDriver_GetClassByName(ByVal bstrClassName As String,?
 ByVal plstClasses As PSIODRIVERLib.IPsEnumVarInfo) As Long
    '
Dim objVarInfo As VarInfo

    If bstrClassName = "Origin" Then

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Country"
        objVarInfo.DisplayName = "Origin Country"
        objVarInfo.Value = "United States"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_ENUM
        objVarInfo.DataDescr = "United States,Puerto Rico"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Postal_Code"
        objVarInfo.DisplayName = "Postal Code"
        objVarInfo.Value = "94588"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
        objVarInfo.DataDescr = "string"
```

```
            objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

            PsIoDriver_GetClassByName = True


        ElseIf bstrTransName = "Destination" Then
            ' Set the data members for this class (code not shown)
            PsIoDriver_GetClassByName = True


        ElseIf bstrTransName = "Package_Information" Then
            ' Set the data members for this class (code not shown)
            PsIoDriver_GetClassByName = True


        Else
            PsIoDriver_GetClassByName = True
        End If

    End Function
```

Since the C++ example uses no classes, GetClassByName returns FALSE.

```
BOOL  SimpleDriver::GetClassByName(const PSIOString& strClassName, ClassDef& classes)
{
   return FALSE;
}
```

Writing the Business Transaction Catalog: GetTransactionByCategory, GetTransactionByName

Categories are used to organize transactions and classes into groups so users can select them more easily. A user designs a Business Interlink definition in the Application Designer, and uses the Business Interlink Search page to search for the transaction or class from which they will create a Business Interlink definition. When they select a category in that page, the transactions or classes that the user can then select are only the transactions or classes that are placed in that <category> tag in the XML design-time plug-in.

Write GetTransactionByCategory to return a list of the transactions under the given category name. There are three default categories for transactions: All Schemas, All Transactions, and Transactions.

The following example shows the transaction names “add numbers” and “concatenate strings” added to the transaction categories.

```
Private Function PsIoDriver_GetTransactionByCategory(ByVal bstrFilter As String,?
 ByVal bstrCriteria As String,?
 ByVal plstTransNames As PSIODRIVERLib.IPsEnumString) As Long
    '
    Dim objPsTransactCategory As PsBstr
    Set objPsTransactCategory = plstTransNames.Add
    objPsTransactCategory.Data = "Calculate Cost"
    Set objPsTransactCategory = plstTransNames.Add
    objPsTransactCategory.Data = "Time-in-Transit"
    Set objPsTransactCategory = plstTransNames.Add
```

```
    objPsTransactCategory.Data = "Tracking"
    PsIoDriver_GetTransactionByCategory = True
    '
End Function
```

Write GetTransactionByName to return a list of the input and output parameters for the given transaction name.

The input parameters are added to the plstInput list, which a variable of type PSIODRIVERLib.IPsEnumVarInfo. Use the Add method to add the input parameters to the plstInput list.

The output parameters are added to the plstOutput list, which a variable of type PSIODRIVERLib.IPsEnumVarInfo. Use the Add method to add the output parameters to the plstOutput list.

The following example shows the add numbers transaction using the Add method to get the input parameters of From, To, and Package_Info, and the output parameter of Service_Rate. It also shows the concatenate strings transaction getting the input parameters of string_1 and string_2, and getting the output parameter of string.

See Chapter 16, “Using The Business Interlink Methods,” Using Parameter Lists, page 245.

```
Private Function PsIoDriver_GetTransactionByName(ByVal bstrTransName As String,?
 ByVal plstInput As PSIODRIVERLib.IPsEnumVarInfo,?
 ByVal plstOutput As PSIODRIVERLib.IPsEnumVarInfo) As Long
    '
    Dim objVarInfo As VarInfo

    If bstrTransName = "Calculate Cost" Then

        Set objVarInfo = plstInput.Add
        objVarInfo.Name = "From"
        objVarInfo.DisplayName = ""
        objVarInfo.Value = ""
        objVarInfo.DataType = ENUM_IOC_DATATYPE_OBJECT
        objVarInfo.DataDescr = "Origin"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstInput.Add
        objVarInfo.Name = "To"
        objVarInfo.DisplayName = ""
        objVarInfo.Value = ""
        objVarInfo.DataType = ENUM_IOC_DATATYPE_OBJECT
        objVarInfo.DataDescr = "Destination"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstInput.Add
        objVarInfo.Name = "Package_Info"
        objVarInfo.DisplayName = ""
        objVarInfo.Value = ""
        objVarInfo.DataType = ENUM_IOC_DATATYPE_OBJECT
        objVarInfo.DataDescr = "Package_Information"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstOutput.Add
```

```
            objVarInfo.Name = "Service_Rate"
            objVarInfo.DisplayName = ""
            objVarInfo.Value = ""
            objVarInfo.DataType = ENUM_IOC_DATATYPE_OBJECT
            objVarInfo.DataDescr = "Service Rate"
            objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

            PsIoDriver_GetTransactionByName = True

        ElseIf bstrTransName = "Tracking" Then
            '  Set the inputs and outputs for this transaction
            PsIoDriver_GetTransactionByName = True

        ElseIf bstrTransName = "Time-in-Transit" Then
            '  Set the inputs and outputs for this transaction
            PsIoDriver_GetTransactionByName = True

        Else
            PsIoDriver_GetTransactionByName = False
        End If
        '
    End Function
```

The following C++ example shows the transaction names “add numbers”, “concatenate strings”, and “machine name” added to the transaction categories.

```
BOOL SimpleDriver::GetTransactionByCategory(const PSIOString& strFilter,?
 const PSIOString& strCriteria, IOSTRINGLIST& transNames)
{
    transNames.push_back(_T("add numbers"));
    transNames.push_back(_T("concatenate strings"));
    transNames.push_back(_T("get machine name"));
    return TRUE; // pTransNames;
}
```

The following example shows the add numbers transaction getting the input parameters of number_1 and number_2, and getting the output parameter of sum. It also shows the concatenate strings transaction getting the input parameters of string_1 and string_2, and getting the output parameter of string.

See Chapter 16, “Using The Business Interlink Methods,” Using Parameter Lists, page 245.

See Chapter 17, “Writing the Design-Time Functionality Using Dynamic Catalog Methods,” AddInput, page 292.

See Chapter 17, “Writing the Design-Time Functionality Using Dynamic Catalog Methods,” AddOutput, page 294.

```
BOOL SimpleDriver::GetTransactionByName(const PSIOString& strTransName,?
 TransactionDef& transDef)
{
    transDef.SetTransactionName(strTransName);

/* Add the input parameters number_1 and number_2, and the output parameter sum,?
 to the add numbers transaction. */
```

```
    if(strTransName == _T("add numbers"))
    {
        transDef.AddInput(VarInfo( _T("number_1"), ?
DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        transDef.AddInput(VarInfo( _T("number_2"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        transDef.AddOutput(VarInfo( _T("sum"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        return TRUE;
    }
/* Add the input parameters string_1 and string_2, and the output parameter string,?
 to the concatenate strings transaction. */
    else if(strTransName == _T("concatenate strings"))
    {
        transDef.AddInput(VarInfo( _T("string_1"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        transDef.AddInput(VarInfo( _T("string_2"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        transDef.AddOutput(VarInfo( _T("string"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        return TRUE;
    }
/* Add the output parameter machine_name to the get machine name transaction. */
    else if(strTransName == _T("get machine name"))
    {
        transDef.AddOutput(VarInfo( _T("machine_name"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        return TRUE;
    }
    else
        return FALSE;
}
```

See Also

Chapter 16, “Using The Business Interlink Methods,” Using Parameter Lists, page 245

Writing a Dynamic Catalog: a Short Example

The previous example code creates a static catalog for the transactions: the number and types of the input and output parameters cannot be changed. To get a dynamic catalog, you need only write these methods to be able to change the number and type of the input and/or output parameters.

For example, you could have the IsSupported method set this plug-in for dynamic catalogs by setting True for ENUM_IOC_DYNAMICCATEGORY instead of for ENUM_IOC_STATICCATEGORY.

```
Private Function PsIoDriver_IsSupported(ByVal eType As?
 PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED) As Long
    '
    Select Case eType
        Case ENUM_IOC_TRANSACTION
            PsIoDriver_IsSupported = True
        Case ENUM_IOC_DYNAMICCATEGORY
            PsIoDriver_IsSupported = True
        Case ENUM_IOC_ISWCHAR
            PsIoDriver_IsSupported = True
        Case Else
            PsIoDriver_IsSupported = False
    End Select
    '
End Function
```

Then write the transaction to be able to add additional input or output parameters, depending upon a condition.

The following example shows GetTransactionByName coded for a transaction that adds two or three input parameters.

Note. When you write your execute method, ExecuteTransaction, make sure it can detect and extract all the input parameters.

```
Private Function PsIoDriver_GetTransactionByName(ByVal bstrTransName As String,?
 ByVal plstInput As PSIODRIVERLib.IPsEnumVarInfo,?
 ByVal plstOutput As PSIODRIVERLib.IPsEnumVarInfo) As Long
    '
    Dim objVarInfo As VarInfo
    If bstrTransName = "add numbers" Then

        Set objVarInfo = plstInput.Add
        objVarInfo.Name = "number_1"
        objVarInfo.DisplayName = ""
        objVarInfo.Value = ""
        objVarInfo.DataType = ENUM_IOC_DATATYPE_INT
        objVarInfo.DataDescr = "int"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
'
        Set objVarInfo = plstInput.Add
        objVarInfo.Name = "number_2"
        objVarInfo.DisplayName = ""
        objVarInfo.Value = ""
        objVarInfo.DataType = ENUM_IOC_DATATYPE_INT
        objVarInfo.DataDescr = "int"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
'
' Add the third input parameter only if a condition is true
```

```
            If (condition) Then
                Set objVarInfo = plstInput.Add
                objVarInfo.Name = "number_3"
                objVarInfo.DisplayName = ""
                objVarInfo.Value = ""
                objVarInfo.DataType = ENUM_IOC_DATATYPE_INT
                objVarInfo.DataDescr = "int"
                objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
            Endif
    '
            Set objVarInfo = plstOutput.Add
            objVarInfo.Name = "sum"
            objVarInfo.DisplayName = "Sum"
            objVarInfo.Value = ""
            objVarInfo.DataType = ENUM_IOC_DATATYPE_INT
            objVarInfo.DataDescr = "int"
            objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

            PsIoDriver_GetTransactionByName = True

        Else
            PsIoDriver_GetTransactionByName = True
        End If
        '
    End Function
```

Using An Example of Design Methods (Visual Basic)

Here is a sample set of methods for a design-time plug-in. It has two transactions to add numbers and concatenate strings. Following this sample is a

- The IsSupported methods shows that the Business Interlink plug-in supports the transaction Interlink type.
- The GetParameterList method provides a string type named Output_File_Name to allow the Business Interlink to output to a file.
- The GetTransactionByCategory method provides the transaction names of add numbers and concatenate strings. The GetTransactionByName provides, for the add numbers transaction, the integer type input parameters number_1 and number_2 and the integer type output parameter sum; and it provides for the concatenate strings transaction the string type input parameters string_1 and string_2 and the string type output parameter string.

```
' GetCategories provides the categories.
Private Function PsIoDriver_GetCategories(?
 ByVal eCatType As PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED,?
 ByVal plstCriterialNames As PSIODRIVERLib.IPsEnumString) As Long
    '
    Dim objPsBstr As PsBstr
    Set objPsBstr = plstCriterialNames.Add
    objPsBstr.Data = "Transactions"
```

```
    Set objPsBstr = plstCriterialNames.Add
    objPsBstr.Data = "All Transactions"
    PsIoDriver_GetCategories = True
    '
End Function

' GetClassByCategory is set to False because this plug-in uses no
' classes except as transaction inputs and outputs.
Private Function PsIoDriver_GetClassByCategory(?
 ByVal bstrFilter As String,?
 ByVal bstrCategory As String,?
 ByVal plstCategory As PSIODRIVERLib.IPsEnumString) As Long
    '
    PsIoDriver_GetClassByCategory = False
    '
End Function

' GetClassByName is set to False because this plug-in uses no
' classes.
Private Function PsIoDriver_GetClassByName(ByVal bstrClassName As String,?
 ByVal plstClasses As PSIODRIVERLib.IPsEnumVarInfo) As Long
    '
Dim objVarInfo As VarInfo

    If bstrClassName = "Origin" Then

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Country"
        objVarInfo.DisplayName = "Origin Country"
        objVarInfo.Value = "United States"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_ENUM
        objVarInfo.DataDescr = "United States,Puerto Rico"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Postal_Code"
        objVarInfo.DisplayName = "Postal Code"
        objVarInfo.Value = "94588"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
        objVarInfo.DataDescr = "string"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        PsIoDriver_GetClassByName = True


    ElseIf bstrTransName = "Destination" Then
        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Country"
        objVarInfo.DisplayName = "Destination Country"
        objVarInfo.Value = "United States"
```

```
        objVarInfo.DataType = ENUM_IOC_DATATYPE_ENUM
        objVarInfo.DataDescr = "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"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Postal_Code"
        objVarInfo.DisplayName = "Postal Code"
        objVarInfo.Value = "10200"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
        objVarInfo.DataDescr = "string"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Address_Type"
        objVarInfo.DisplayName = "Address Type"
        objVarInfo.Value = "Commercial"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_ENUM
        objVarInfo.DataDescr = "Commercial,Residential"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        PsIoDriver_GetClassByName = True


   ElseIf bstrTransName = "Package_Information" Then
        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Drop_off_Pickup"
        objVarInfo.DisplayName = "Drop-off Pickup"
        objVarInfo.Value = "One Time Pickup"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_ENUM
        objVarInfo.DataDescr = "Regular Daily Pickup,On Call Air,One Time Pickup,?
Letter Center,Customer Counter"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Packaging"
        objVarInfo.DisplayName = "Packaging"
        objVarInfo.Value = "Express Box"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_ENUM
        objVarInfo.DataDescr = "Your Packaging,Letter Envelop,Tube,?
Express Box,Worldwide 25KG Box,Worldwide 10KG Box"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Weight"
        objVarInfo.DisplayName = "Weight"
        objVarInfo.Value = "1"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_INT
        objVarInfo.DataDescr = "int"
```

```
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Length"
        objVarInfo.DisplayName = "Length"
        objVarInfo.Value = "4"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_INT
        objVarInfo.DataDescr = "int"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Width"
        objVarInfo.DisplayName = "Length"
        objVarInfo.Value = "4"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_INT
        objVarInfo.DataDescr = "int"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Height"
        objVarInfo.DisplayName = "Length"
        objVarInfo.Value = "4"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_INT
        objVarInfo.DataDescr = "int"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        PsIoDriver_GetClassByName = True


    ElseIf bstrTransName = "Service_Rate" Then
        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Service_Type"
        objVarInfo.DisplayName = "Service Type"
        objVarInfo.Value = "Next Day Air Early AM"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
        objVarInfo.DataDescr = "string"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Guaranteed_By"
        objVarInfo.DisplayName = "Guaranteed By"
        objVarInfo.Value = "8:00 AM Next Day"
        objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
        objVarInfo.DataDescr = "string"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstClasses.Add
        objVarInfo.Name = "Rate"
        objVarInfo.DisplayName = "Rate"
        objVarInfo.Value = "50:00"
```

```
        objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
        objVarInfo.DataDescr = "string"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        PsIoDriver_GetClassByName = True


    Else
        PsIoDriver_GetClassByName = True
    End If


End Function

' GetCriteriaLogicalOperators and GetCriteriaRelationalOperators are
' set to False because this plug-in is not a query or update type.
Private Function PsIoDriver_GetCriteriaLogicalOperators(ByVal plstOperators?
 As PSIODRIVERLib.IPsEnumString) As Long
    '
    PsIoDriver_GetCriteriaLogicalOperators = False
    '
End Function
Private Function PsIoDriver_GetCriteriaRelationalOperators(ByVal bstrType As String,?
 ByVal plstOperators As PSIODRIVERLib.IPsEnumString) As Long
    '
    PsIoDriver_GetCriteriaRelationalOperators = False
    '
End Function

' GetDesc provides a short text description of this plug-in.
Private Function PsIoDriver_GetDesc() As String
    '
    PsIoDriver_GetDesc = "Freight Carrier"
    '
End Function

' GetLastErrorMessage provides a blank error message.
Private Sub PsIoDriver_GetLastErrorMessage(ByVal pDrvMsg As?
 PSIODRIVERLib.IPsDriverMessage)
    '
    pDrvMsg.Message = ""
    pDrvMsg.MessageGroupId = 0
    pDrvMsg.MessageId = 0
    '
End Sub

' Provides a configuration parameter named Output_File_Name.
Private Function PsIoDriver_GetParameterList(ByVal plstParams As?
 PSIODRIVERLib.IPsEnumVarInfo) As Long
    '
```

```
    Dim objVarInfo As VarInfo
    Set objVarInfo = plstParams.Add
    objVarInfo.Name = "URL"
    objVarInfo.DisplayName = ""
    objVarInfo.Value = "file://PSCustomer.dll"
    objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
    objVarInfo.DataDescr = "string"
    objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
    PsIoDriver_GetParameterList = True
    '
End Function

' Provides the transaction names
Private Function PsIoDriver_GetTransactionByCategory(ByVal bstrFilter As String,?
 ByVal bstrCriteria As String,?
 ByVal plstTransNames As PSIODRIVERLib.IPsEnumString) As Long
    '
    Dim objPsTransactCategory As PsBstr
    Set objPsTransactCategory = plstTransNames.Add
    objPsTransactCategory.Data = "Calculate Cost"
    Set objPsTransactCategory = plstTransNames.Add
    objPsTransactCategory.Data = "Time-in-Transit"
    Set objPsTransactCategory = plstTransNames.Add
    objPsTransactCategory.Data = "Tracking"
    PsIoDriver_GetTransactionByCategory = True
    '
End Function

' Provides the input and output parameters for the transactions.
Private Function PsIoDriver_GetTransactionByName(ByVal bstrTransName As String,?
 ByVal plstInput As PSIODRIVERLib.IPsEnumVarInfo,?
 ByVal plstOutput As PSIODRIVERLib.IPsEnumVarInfo) As Long
    '
    Dim objVarInfo As VarInfo

    If bstrTransName = "Calculate Cost" Then

        Set objVarInfo = plstInput.Add
        objVarInfo.Name = "From"
        objVarInfo.DisplayName = ""
        objVarInfo.Value = ""
        objVarInfo.DataType = ENUM_IOC_DATATYPE_OBJECT
        objVarInfo.DataDescr = "Origin"
        objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

        Set objVarInfo = plstInput.Add
        objVarInfo.Name = "To"
        objVarInfo.DisplayName = ""
        objVarInfo.Value = ""
        objVarInfo.DataType = ENUM_IOC_DATATYPE_OBJECT
```

```
            objVarInfo.DataDescr = "Destination"
            objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

            Set objVarInfo = plstInput.Add
            objVarInfo.Name = "Package_Info"
            objVarInfo.DisplayName = ""
            objVarInfo.Value = ""
            objVarInfo.DataType = ENUM_IOC_DATATYPE_OBJECT
            objVarInfo.DataDescr = "Package_Information"
            objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

            Set objVarInfo = plstOutput.Add
            objVarInfo.Name = "Service_Rate"
            objVarInfo.DisplayName = ""
            objVarInfo.Value = ""
            objVarInfo.DataType = ENUM_IOC_DATATYPE_OBJECT
            objVarInfo.DataDescr = "Service Rate"
            objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

            PsIoDriver_GetTransactionByName = True

        ElseIf bstrTransName = "Tracking" Then
            '  Set the inputs and outputs for this transaction
            PsIoDriver_GetTransactionByName = True

        ElseIf bstrTransName = "Time-in-Transit" Then
            '  Set the inputs and outputs for this transaction
            PsIoDriver_GetTransactionByName = True

        Else
            PsIoDriver_GetTransactionByName = False
        End If
        '
    End Function

    ' Provides a version number for this plug-in.
    Private Function PsIoDriver_GetVer() As String
        '
        PsIoDriver_GetVer = "1.0.0"
        '
    End Function

    ' Sets the Business Interlink type to transaction, static
    ' categories.
    Private Function PsIoDriver_IsSupported(ByVal eType As?
     PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED) As Long
        '
        Select Case eType
            Case ENUM_IOC_TRANSACTION
                PsIoDriver_IsSupported = True
```

```
        Case ENUM_IOC_STATICCATEGORY
            PsIoDriver_IsSupported = True
        Case ENUM_IOC_ISWCHAR
            PsIoDriver_IsSupported = True
        Case Else
            PsIoDriver_IsSupported = False
    End Select
    '
End Function


Private Function PsIoDriver_SetCategoryType(ByVal eCatType As?
 PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED) As Long
    '
    PsIoDriver_SetCategoryType = False
    '
End Function
Private Function PsIoDriver_SetParameterList(ByVal plstParams As?
 PSIODRIVERLib.IPsEnumVarInfo) As Long
    '
    PsIoDriver_SetParameterList = False
    '
End Function
```

Using An Example of Design Methods (C++)

Here is a sample set of methods for a design-time plug-in. It has two transactions to add numbers and concatenate strings. Following this sample is a

- The IsSupported methods shows that the Business Interlink plug-in supports the transaction Interlink type.
- The GetParameterList method provides a string type named Output_File_Name to allow the Business Interlink to output to a file.
- The GetTransactionByCategory method provides the transaction names of add numbers and concatenate strings. The GetTransactionByName provides, for the add numbers transaction, the integer type input parameters number_1 and number_2 and the integer type output parameter sum; and it provides for the concatenate strings transaction the string type input parameters string_1 and string_2 and the string type output parameter string.

```
const TCHAR * SimpleDriver::GetDesc() const
{
return _T("Simple interface driver: adds 2 #, concats 2 strs");
}

const TCHAR * SimpleDriver::GetVersion() const
{
return _T("1.00");
}
```

```
BOOL SimpleDriver::IsSupported(EIOCINTERFACESUPPORTED option)
{
   switch(option)
   {
      case IOC_TRANSACTION:
      case IOC_STATICCATEGORY:
      return TRUE;

      case IOC_ISWCHAR:
      return  (sizeof(TCHAR) != sizeof(char));

      default:
      return FALSE;
   }
}

// Create the transaction categories.
BOOL SimpleDriver::GetCategories(EIOCINTERFACESUPPORTED type,?
 IOSTRINGLIST& criteria1Names)
{
if(type == IOC_TRANSACTION)
{
   criteria1Names.push_back(_T("Transactions"));
   criteria1Names.push_back(_T("All Transactions"));
}
return TRUE;
}

// Create the transaction names.
BOOL SimpleDriver::GetTransactionByCategory(const PSIOString& strFilter,?
 const PSIOString& strCriteria, IOSTRINGLIST& transNames)
{
    transNames.push_back(_T("add numbers"));
    transNames.push_back(_T("concatenate strings"));
    transNames.push_back(_T("get machine name"));
    return TRUE; // pTransNames;
}

BOOL SimpleDriver::GetTransactionByName(const PSIOString& strTransName,?
 TransactionDef& transDef)
{
    transDef.SetTransactionName(strTransName);

/* Add the input parameters number_1 and number_2, and the output parameter sum,?
 to the add numbers transaction. */
    if(strTransName == _T("add numbers"))
    {
        transDef.AddInput(VarInfo( _T("number_1"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
```

```
 eVAR_ATTR_REQUIRED));
        transDef.AddInput(VarInfo( _T("number_2"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        transDef.AddOutput(VarInfo( _T("sum"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        return TRUE;
    }
/* Add the input parameters string_1 and string_2, and the output parameter string,?
 to the concatenate strings transaction. */
    else if(strTransName == _T("concatenate strings"))
    {
        transDef.AddInput(VarInfo( _T("string_1"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        transDef.AddInput(VarInfo( _T("string_2"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        transDef.AddOutput(VarInfo( _T("string"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        return TRUE;
    }
/* Add the output parameter machine_name to the get machine name transaction. */
    else if(strTransName == _T("get machine name"))
    {
       transDef.AddOutput(VarInfo( _T("machine_name"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
        return TRUE;
    }
    else
        return FALSE;
}



BOOL  SimpleDriver::GetClassByCategory(const PSIOString& strFilter,?
 const PSIOString& Category, IOSTRINGLIST& list)
{
        return FALSE;
}


BOOL  SimpleDriver::GetClassByName(const PSIOString& strClassName,?
 ClassDef& classes)
{
        return FALSE;
```

```
}


BOOL  SimpleDriver::GetCriteriaRelationalOperators(?
 const PSIOString& strType, IOSTRINGLIST& list)
{
        return FALSE;
}

BOOL  SimpleDriver::GetCriteriaLogicalOperators(IOSTRINGLIST& list)
{
        return FALSE;
}


VarInfo config(_T("Output_File_Name"), DataType(IOC_DATATYPE_STRING,?
 _T("")), _T(""), _T(""), _T(""), eVAR_ATTR_NONE);

BOOL  SimpleDriver::GetParameterList(VARINFOLIST& list)
{
        list.push_back(&config);
        return TRUE;
}
```

Dynamic Catalog Class Methods

When you do not use an XML design-time plug-in, you write the design-time functionality using the dynamic catalog class methods. The class for these methods is the class for your Interlink Object.

FetchNextChunk, PsIoDriver_FetchNextChunk

Syntax

C++:

```
virtual BOOL FetchNextChunk(InterfaceObject *);
```

Visual Basic:

```
FetchNextChunk(IPsEnumString) As Long
```

Description

Write this method to return the next chunk of the input or output table.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
IpsEnumString	The table from which a chunk is being returned.
InterfaceObject *	The interlink object.

Returns

True if a chunk was returned, false otherwise.

Value	Meaning
BOOL Long	True if a chunk was returned, false otherwise.

GetCategories, PsIoDriver_GetCategories

Syntax

C++:

```
virtual BOOL GetCategories(EIOCINTERFACESUPPORTED type, IOSTRINGLIST& list) = 0;
```

Visual Basic:

```
GetCategories(type As ENUM_EIOCINTERFACESUPPORTED, list As IPsEnumString) As Long
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this method to return a list of categories for a given interface type: transactions or classes. This is used in the Business Interlink Search page. Since GetCategories is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
Type	The interface type for which to list categories.
List	The list of categories that is returned.

Returns

True if a list of categories is returned, False otherwise.

Value	Meaning
BOOL Long	True if a list of categories is returned, False otherwise.
List	A list of strings containing the categories.

Example

The following C++ example sets the categories to be Transactions and All Transactions.

```
BOOL SimpleDriver::GetCategories(EIOCINTERFACESUPPORTED type,?
 IOSTRINGLIST& criteria1Names)
{
if(type == IOC_TRANSACTION)
{
   criteria1Names.push_back(_T("Transactions"));
   criteria1Names.push_back(_T("All Transactions"));
}
return TRUE;
}
```

The following Visual Basic example sets the categories to be Transactions and All Transactions.

```
Private Function PsIoDriver_GetCategories(?
 ByVal eCatType As PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED,?
 ByVal plstCriterialNames As PSIODRIVERLib.IPsEnumString) As Long
    '
    Dim objPsBstr As PsBstr
    Set objPsBstr = plstCriterialNames.Add
    objPsBstr.Data = "Transactions"
    Set objPsBstr = plstCriterialNames.Add
    objPsBstr.Data = "All Transactions"
    PsIoDriver_GetCategories = True
    '
End Function
```

GetClassByName, PsIoDriver_GetClassByName

Syntax

C++:

```
virtual BOOL GetClassByName(const PSIOString& strClassName, ClassDef& classes) = 0;
```

Visual Basic:

```
PsIoDriver_GetClassByName(ByVal strClassName As String, ByVal classes As
PSIODRIVERLib.IPsEnumVarInfo) As Long
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this methods to return a list of the class data members for the class named *classes*. Since GetClassByName is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
strClassName	A character string. GetClassByName finds all the class names containing this character string.
classes	An array of strings returned that contains the data members for the class named strClassName.

Returns

True if a class data member list was returned, False otherwise.

Value	Meaning
classes	An array of strings that contains the data members for the class named *strClassName.*
BOOL, Long	True if a class data member list was returned, False otherwise.

Example

The following C++ example adds the data members named sender, cc, subject, date, and body to the class named Emessage. The plug-in is an email plug-in. Call the SetClassName method to set the class name, and use the AddDataMemberDef to add the data members to the class.

See Chapter 17, "Writing the Design-Time Functionality Using Dynamic Catalog Methods," SetClassName, page 296.

See Chapter 17, "Writing the Design-Time Functionality Using Dynamic Catalog Methods," AddDataMemberDef, page 293.

```
BOOL  EmailDriver::GetClassByName(const PSIOString& strClassName, ClassDef& classes)
{
   if(strClassName == "Emessage")
   {
      classes.SetClassName(strClassName);
      classes.AddDataMemberDef(VarInfo(_T("sender"),?
```

```
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("cc"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("subject"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("date"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("body"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      return TRUE;
    }
    return FALSE;
}
```

Visual Basic:

The following Visual Basic example adds the data members named sender, cc, subject, date, and body to the class named Emessage. The plug-in is an email plug-in. Use the Add method to add the data members to the class.

See Chapter 17, "Writing the Design-Time Functionality Using Dynamic Catalog Methods," Add, page 289.

```
Private Function PsIoDriver_GetClassByName(ByVal bstrClassName As String,?
 ByVal plstClasses As PSIODRIVERLib.IPsEnumVarInfo) As Long
'
Dim objVarInfo As VarInfo

If bstrClassName = "Emessage" Then
'
   Set objVarInfo = plstClasses.Add
   objVarInfo.Name = "sender"
   objVarInfo.DisplayName = ""
   objVarInfo.Value = ""
   objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
   objVarInfo.DataDescr = "sender"
   objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
'
   Set objVarInfo = plstClasses.Add
   objVarInfo.Name = "cc"
   objVarInfo.DisplayName = ""
   objVarInfo.Value = ""
   objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
   objVarInfo.DataDescr = "cc"
   objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
'
   Set objVarInfo = plstClasses.Add
   objVarInfo.Name = "subject"
   objVarInfo.DisplayName = ""
```

```
   objVarInfo.Value = ""
   objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
   objVarInfo.DataDescr = "subject"
   objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
'
   Set objVarInfo = plstClasses.Add
   objVarInfo.Name = "date"
   objVarInfo.DisplayName = ""
   objVarInfo.Value = ""
   objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
   objVarInfo.DataDescr = "date"
   objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
'
   Set objVarInfo = plstClasses.Add
   objVarInfo.Name = "body"
   objVarInfo.DisplayName = ""
   objVarInfo.Value = ""
   objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
   objVarInfo.DataDescr = "body"
   objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
'
   PsIoDriver_GetClassByName = True
'
Else
   PsIoDriver_GetClassByName = False
End If
```

GetCriteriaRelationalOperators, PsIoDriver_GetCriteriaRelationalOperators

Syntax

C++:

```
virtual BOOL GetCriteriaRelationalOperators(const PSIOString& strType,
IOSTRINGLIST& list) = 0;
```

Visual Basic:

```
PsIoDriver_GetCriteriaRelationalOperators(ByVal strType As String,
list As IPsEnumString) As Long
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this method to return a list of all the relational operators for queries. The server knows what actions to perform for each operator; this method tells the server what action is performed for what symbol. Since GetCriteriaRelationalOperators is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
StrType	A character string.
List	An array of strings returned that contains the relational operators for the data member type strType.

Returns

True if a relational operator list was returned, False otherwise.

Value	Meaning
List	An array of strings that contains the relational operators for the data member type *strType*.
BOOL Long	True if a relational operator list was returned, False otherwise.

Example

The following C++ example creates the relational operators of =, >, <, and <>.

```
BOOL RecordDriver::GetCriteriaRelationalOperators(const PSIOString& strType,?
 IOSTRINGLIST& alist)
{
   alist.push_back(_T("="));
   alist.push_back(_T(">"));
   alist.push_back(_T("<"));
   alist.push_back(_T("<>"));
   return TRUE;
}
```

The following Visual Basic example creates the relational operators of =, >, <, and <>.

```
Private Function PsIoDriver_GetCriteriaRelationalOperators(ByVal bstrType As String,?
 ByVal plstOperators As PSIODRIVERLib.IPsEnumString) As Long
    '
    Dim objPsBstr As PsBstr
    Set objPsBstr = plstOperators.Add
    objPsBstr.Data = "="
    Set objPsBstr = plstOperators.Add
    objPsBstr.Data = ">"
    Set objPsBstr = plstOperators.Add
    objPsBstr.Data = "<"
    Set objPsBstr = plstOperators.Add
```

```
        objPsBstr.Data = "<>"
        PsIoDriver_GetCriteriaRelationalOperators = True
        '
    End Function
```

GetCriteriaLogicalOperators, PsIoDriver_ GetCriteriaLogicalOperators

Syntax

C++:

```
virtual IOSTRINGLIST GetCriteriaLogicalOperators(const PSIOString& strType) = 0;
```

Visual Basic:

```
PsIoDriver_GetCriteriaLogicalOperators(ByVal plstOperators As
 PSIODRIVERLib.IPsEnumString) As Long
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this method to return a list of all the logical operators for queries. The server knows what actions to perform for each operator; this method tells the server what action is performed for what symbol. Since GetCriteriaLogicalOperators is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
list	An array of strings returned that contains the logical operators for queries.

Returns

True if a logical operator list was returned, False otherwise.

Value	Meaning
list	An array of strings that contains the logical operators for queries.
BOOL	True if a logical operator list was returned, False otherwise.

Example

The following C++ example creates the logical operators of AND, OR, and NOT.

```
BOOL  RecordDriver::GetCriteriaLogicalOperators(IOSTRINGLIST& alist)
{
   alist.push_back(_T("AND"));
   alist.push_back(_T("OR"));
   alist.push_back(_T("NOT"));
   return TRUE;
}
```

The following Visual Basic example creates the logical operators of AND, OR, and NOT.

```
Private Function PsIoDriver_GetCriteriaLogicalOperators(ByVal bstrType As String,?
 ByVal plstOperators As PSIODRIVERLib.IPsEnumString) As Long
    '
    Dim objPsBstr As PsBstr
    Set objPsBstr = plstOperators.Add
    objPsBstr.Data = "AND"
    Set objPsBstr = plstOperators.Add
    objPsBstr.Data = "OR"
    Set objPsBstr = plstOperators.Add
    objPsBstr.Data = "NOT"
    PsIoDriver_GetCriteriaLogicalOperators = True
    '
End Function
```

GetDesc, PsIoDriver_GetDesc

Syntax

C++:

```
virtual const TCHAR * GetDesc() = 0;
```

Visual Basic:

```
PsIoDriver_GetDesc() As String
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this method to return a string containing the description of this Business Interlink. This string is used in the New Business Interlink page. Since GetDesc is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

None.

Returns

The description of this Business Interlink.

Value	Meaning
TCHAR*, String.	Contains the description of this Business Interlink.

Example

The following C++ example creates the description for a plug-in that adds numbers and concatenates strings.

```
const TCHAR * SimpleDriver::GetDesc() const
{
return _T("Simple interface plug-in: adds 2 #, concats 2 strs");
}
```

The following Visual Basic example creates the description for a plug-in that adds numbers and concatenates strings.

```
Private Function PsIoDriver_GetDesc() As String
    '
    PsIoDriver_GetDesc =
       "Simple interface plug-in: adds 2 #, concats 2 strs"
    '
End Function
```

GetLastErrorMessage

Syntax

C++:

```
virtual const IODrvrMsg& GetLastErrorMessage () { return m_ErrMsg; }
```

Visual Basic:

```
GetLastErrorMessage(pDrvMsg As IPsDriverMessage)
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this method to return the latest error message.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

None.

Returns

The error message.

Value	Meaning
m_ErrMsg, pDrvMsg	The returned error message.

GetClassByCategory, PsIoDriver_GetClassByCategory

Syntax

C++:

```
virtual BOOL GetClassByCategory(const PSIOString& strFilter, const PSIOString&
strCategory, IOSTRINGLIST& list) = 0;
```

Visual Basic:

```
PsIoDriver_GetClassByCategory(strFilter As String, strCategory As String,
list As IPsEnumString) As Long
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this method to return a list of class names based upon the given *Category*. This is used in the Interface Definition Search page. Since GetClassByCategory is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
strFilter	A string that you can search on, such as *sales* to find all transactions/classes with *sales* in their names.
strCategory	A string containing the category.
list	The array of strings returned that contains the class names (or descriptions) in this category.

Returns

True if an array is returned, false otherwise.

Value	Meaning
list	The list of class names (or descriptions) for the given category.
BOOL, Long	True if an array is returned, false otherwise.

Example

The following C++ example creates a list of two class names: class1 and class2. It places class1 into category1 and class1 and class2 into category2.

```
BOOL SimpleDriver::GetClassByCategory
   (const PSIOString& strFilter,
    const PSIOString& strCategory,
    IOSTRINGLIST& classNames)
{
   if (strCategory == _T("category1"))
   {
      classNames.push_back(_T("class1"));
      return TRUE;
   }
   if (strCategory == _T("category2"))
   {
      classNames.push_back(_T("class1"));
      classNames.push_back(_T("class2"));
      return TRUE;
   }
return FALSE; //
}
```

The following Visual Basic example creates a list of two class names: class1 and class2.

```
Private Function PsIoDriver_GetClassByCategory
   (ByVal strFilter As String,
    ByVal strCriteria As String,
    ByVal plstClassNames As PSIODRIVERLib.IPsEnumString) As Long
   '
   Dim objPsClassCategory As PsBstr
   If (strCriteria = "category1" Then
      Set objPsClassCategory = plstClassNames.Add
      objPsClassCategory.Data = "class1"
      PsIoDriver_GetClassByCategory = True
   ElseIf (strCriteria = "category2" Then
      Set objPsClassCategory = plstClassNames.Add
      objPsClassCategory.Data = "class1"
      Set objPsClassCategory = plstClassNames.Add
      objPsClassCategory.Data = "class2"
      PsIoDriver_GetClassByCategory = True
   Else
```

```
        PsIoDriver_GetClassByCategory = False
    End If
    '
End Function
```

GetParameterList, PsIoDriver_GetParameterList

Syntax

C++:

```
virtual BOOL GetParameterList(VARINFOLIST& list) = 0;
```

Visual Basic:

```
PsIoDriver_GetParameterList(list As IPsEnumVarInfo) As Long
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this method to return an array containing the data types and names of the configuration parameters for this system. Since GetParameterList is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
list	An array of strings containing the configuration parameters.

Returns

True if an array is returned, false otherwise.

Value	Meaning
list	The list of configuration parameters.
BOOL	True if an array is returned, false otherwise.

Example

The following C++ example creates a configuration parameter named Output_File_Name. Use the push_back method to create the parameter.

Each push_back call creates a data member containing the following information:

- A string containing the data member name.

- The data type of the variable.
- A string containing a class name if the DataType is IOC_DATATYPE_OBJECT or IOC_DATATYPE_LIST_OBJECT.
- A string containing the default value, if any.
- A bool (TRUE or FALSE): whether or not this data member is required. eVAR_ATTR_NONE means not required, eVAR_ATTR_REQUIRED means required.

```
BOOL  SimpleDriver::GetParameterList(VARINFOLIST& list)
{
   list.push_back(new VarInfo(_T("Output_File_Name"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_NONE));
   return TRUE;
}
```

Visual Basic:

The following Visual Basic example creates a configuration parameter named Output_File_ Name. Use the Add method to add the parameter.

See Chapter 16, “Using The Business Interlink Methods,” Using Parameter Lists, page 245.

See Chapter 17, “Writing the Design-Time Functionality Using Dynamic Catalog Methods,” Add, page 289.

```
Private Function PsIoDriver_GetParameterList(ByVal plstParams As?
 PSIODRIVERLib.IPsEnumVarInfo) As Long
   '
   Dim objVarInfo As VarInfo
   Set objVarInfo = plstParams.Add
   objVarInfo.Name = "Output_File_Name"
   objVarInfo.DisplayName = ""
   objVarInfo.Value = ""
   objVarInfo.DataType = ENUM_IOC_DATATYPE_STRING
   objVarInfo.Attribute = ENUM_VAR_ATTR_NONE
   PsIoDriver_GetParameterList = True
   '
End Function
```

GetTransactionByCategory, PsIoDriver_ GetTransactionByCategory

Syntax

C++:

```
virtual IOSTRINGLIST * GetTransactionByCategory(const PSIOString& strFilter, const
PSIOString& Category, IOSTRINGLIST& transNames) = 0;
```

Visual Basic:

```
PsIoDriver_GetTransactionByCategory(strFilter As String, bstrCriteria As
String, TransNames As IPsEnumString) As Long
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this method to return a list of transaction names based upon the given *Category*. This is used in the Interface Definition Search page. Since GetTransactionByCategory is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
strFilter	A string that you can search on, such as *sales* to find all transactions/classes with *sales* in their names.
strCriteria	A string containing the category.
transNames	The array of strings returned that contains the transaction names (or descriptions) in this category.

Returns

True if an array is returned, false otherwise.

Value	Meaning
transNames	The list of transaction names (or descriptions) for the given category.
BOOL, Long	True if an array is returned, false otherwise.

Example

The following C++ example creates a list of two transaction names: add numbers and concatenate strings.

See Chapter 17, “Writing the Design-Time Functionality Using Dynamic Catalog Methods,” push_back, page 288.

```
BOOL SimpleDriver::GetTransactionByCategory
   (const PSIOString& strFilter,
    const PSIOString& strCriteria,
    IOSTRINGLIST& transNames)
{
   if (strCategory == _T("category1"))
   {
      transNames.push_back(_T("add numbers"));
```

```
      return TRUE;
   }
   if (strCategory == _T("category2"))
   {
      transNames.push_back(_T("add numbers"));
      transNames.push_back(_T("concatenate strings"));
      return TRUE;
   }
return FALSE; //
}
```

The following Visual Basic example creates a list of two transaction names: add numbers and concatenate strings.

```
Private Function PsIoDriver_GetTransactionByCategory
   (ByVal bstrFilter As String,
    ByVal bstrCriteria As String,
    ByVal plstTransNames As PSIODRIVERLib.IPsEnumString) As Long
   '
   Dim objPsTransactCategory As PsBstr
   If (strCriteria = "category1" Then
      Set objPsTransactCategory = plstTransNames.Add
      objPsTransactCategory.Data = "add numbers"
      PsIoDriver_GetTransactionByCategory = True
   ElseIf (strCriteria = "category2" Then
      Set objPsTransactCategory = plstTransNames.Add
      objPsTransactCategory.Data = "add numbers"
      Set objPsTransactCategory = plstTransNames.Add
      objPsTransactCategory.Data = "concatenate strings"
      PsIoDriver_GetTransactionByCategory = True
   Else
      PsIoDriver_GetTransactionByCategory = False
   End If
   '
End Function
```

GetTransactionByName, PsIoDriver_GetTransactionByName

Syntax

C++:

```
virtual BOOL GetTransactionByName(const PSIOString& strTransName,
TransactionDef& transactions) = 0;
```

Visual Basic:

```
PsIoDriver_GetTransactionByName(strTransName As String, plstInput As IPsEnumVarInfo,
plstOutput As IPsEnumVarInfo) As Long
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

Write this method to return a list of the names and data types of the input/output parameters for the transaction named *strTransName*. Since GetTransactionByName is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
strTransName	A character string. GetTransactionByName finds all the transaction names containing this character string.
TransDef	The TransactionDef class. You will set an array of strings in that class that contains the input/output parameters for the transaction named strTransName.
PlstInput	

Returns

True if a transaction parameter list was returned, False otherwise.

Value	Meaning
TransDef	The array of strings that contains the input/output parameters for the transaction named *strTransName.*
BOOL, long	True if a transaction parameter list was returned, False otherwise.

Example

The following C++ example sets the input and output parameters for the transaction named add numbers and concatenate strings. For add numbers, the input parameters are number_1 and number_2, and the output parameter is sum. For concatenate strings, the input parameters are string_1 and string_2, and the output parameter is string. Use the AddInput method to add the input parameters, and use the method AddOutput to add the output parameters.

```
BOOL SimpleDriver::GetTransactionByName(const PSIOString& strTransName,?
 TransactionDef& transDef)
{
   transDef.SetTransactionName(strTransName);

   if(strTransName == _T("add numbers"))
   {
      transDef.AddInput(VarInfo( _T("number_1"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      transDef.AddInput(VarInfo( _T("number_2"),?
```

```
   DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
   eVAR_ATTR_REQUIRED));
        transDef.AddOutput(VarInfo( _T("sum"),?
   DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
   eVAR_ATTR_REQUIRED));
        return TRUE;
      }
      else if(strTransName == _T("concatenate strings"))
      {
         transDef.AddInput(VarInfo( _T("string_1"),?
   DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
   eVAR_ATTR_REQUIRED));
         transDef.AddInput(VarInfo( _T("string_2"),?
   DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
   eVAR_ATTR_REQUIRED));
         transDef.AddOutput(VarInfo( _T("string"),?
   DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
   eVAR_ATTR_REQUIRED));
         return TRUE;
      }
      else
         return FALSE;
  }
```

Visual Basic:

The following Visual Basic example sets the input and output parameters for the transaction named add numbers and concatenate strings. For add numbers, the input parameters are number_1 and number_2, and the output parameter is sum. For concatenate strings, the input parameters are string_1 and string_2, and the output parameter is string. Use the Add method to add the input and output parameter information.

IsSupported, PsIoDriver_IsSupported

Syntax

C++:

```
virtual BOOL IsSupported(EIOCINTERFACETYPE type) = 0;
```

Visual Basic:

```
PsIoDriver_IsSupported(ByVal Type As PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED) As Long
```

Description

Write this method to return true or false: true if the given type is a supported interface type, false otherwise. Since IsSupported is a virtual method, you must write the code for this method.

The classes are:

- C++: DLLBaseDriver
- Visual Basic: PsIoDriver

Parameters

Parameter	Description
type	the interface type to test. The allowed values are:
	C++: IOC_UNKNOWN (0), IOC_OBJQUERY (1), IOC_TRANSACTION (2), IOC_OBJADD (3), IOC_OBJUPDATE (4), IOC_OBJDELETE (5), IOC_ORDERBYASC(6), IOC_ORDERBYDESC(7), IOC_INPUT_CLASSEXPANSION(8), IOC_STAICCATEGORY(9), IOC_DYNAMICCATEGORY(10), IOC_ISWCHAR(11)
	Visual Basic: ENUM_IOC_UNKNOWN (0), ENUM_IOC_OBJQUERY (1), ENUM_IOC_TRANSACTION (2), ENUM_IOC_OBJADD (3), ENUM_IOC_OBJUPDATE (4), ENUM_IOC_OBJDELETE (5), ENUM_IOC_ORDERBYASC(6), ENUM_IOC_ORDERBYDESC(7), ENUM_IOC_INPUT_CLASSEXPANSION(8), ENUM_IOC_STATICCATEGORY(9), ENUM_IOC_DYNAMICCATEGORY(10), ENUM_IOC_ISWCHAR(11)

Returns

True if the given type is a supported interface type, False otherwise.

Value	Meaning
BOOL, Long	True if the given type is a supported interface type, False otherwise.

Example

The following C++ example sets the supported types for the SimpleDriver Business Interlink to support transactions, and to set the categories to static (the input/output parameters for the transaction are not changeable in data type or number of parameters).

```
BOOL SimpleDriver::IsSupported(EIOCINTERFACESUPPORTED option)
{
   switch(option)
   {
      case IOC_TRANSACTION:
      case IOC_STATICCATEGORY:
      return TRUE;

      case IOC_ISWCHAR:
      return  (sizeof(TCHAR) != sizeof(char));

      default:
      return FALSE;
   }
}
```

The following Visual Basic example sets the supported types for the SimpleDriver Business Interlink to support transactions, and to set the categories to static (the input/output parameters for the transaction are not changeable in data type or number of parameters).

```
Private Function PsIoDriver_IsSupported(ByVal eType As?
 PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED) As Long
    '
    Select Case eType
        Case ENUM_IOC_TRANSACTION
            PsIoDriver_IsSupported = True
        Case ENUM_IOC_STATICCATEGORY
            PsIoDriver_IsSupported = True
        Case ENUM_IOC_ISWCHAR
            PsIoDriver_IsSupported = True
        Case Else
            PsIoDriver_IsSupported = False
    End Select
    '
End Function
```

push_back

Syntax

C++:

Class: PSIOStringList

```
void push_back(PSIOString);
void push_back(char *);
```

Class: PSIOVarInfoList

```
void push_back(VarInfo *);
```

Description

Pushes data into a parameter.

Parameters

None.

Returns

None.

Example

The following C++ code pushes a transaction category.

```
BOOL SimpleDriver::GetCategories(EIOCINTERFACESUPPORTED type,?
 IOSTRINGLIST& criteria1Names)
{
if(type == IOC_TRANSACTION)
{
   criteria1Names.push_back(_T("Transactions"));
```

```
   criteria1Names.push_back(_T("All Transactions"));
}
return TRUE;
}
```

The following C++ code pushes a configuration parameter.

```
BOOL  SimpleDriver::GetParameterList(VARINFOLIST& list)
{
   list.push_back(new VarInfo(_T("Output_File_Name"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_NONE));
   return TRUE;
}
```

Adding Input/Output Parameters to Transactions and Data Members to Classes with Transaction/Class Methods

This section provides an overview of the transaction and class methods and discusses each method.

Transaction and Class Methods

You use the Transaction and Class method when you write the design-time functionality using the dynamic catalog class methods, instead of using an XML design-time plug-in. The class for these methods is the class for your Interlink Object. These methods allow you to set information when you design the Transactions and for Classes for your Business Interlink plug-in.

For C++, you must include the header file ioutil.h for this class.

The following list shows which methods you use to operate on the input and output tables:

- Use Add to add a parameter to a transaction or a data member to a class in Visual Basic.
- Use AddDataMemberDef to add a data member to a class in C++.
- Use AddInput to add an input parameter to a transaction in C++.
- Use AddOutput to add an output parameter to a transaction in C++.
- Use SetClassName to set a class name in C++.
- Use SetTransactionName to set a transaction name in C++.

Add

Syntax

Visual Basic:

```
Add()
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

The class is IPsEnumVarInfo.

The Visual Basic **Add** method adds a configuration parameter, an input or output parameter to a transaction, a data member to a class, or a category name to the category list for the transactions or classes. When adding parameters or data members, you supply the following information:

- A string containing the parameter or data member name.
- The data type of the variable.
- A string containing a display name. This name shows in the Input Name and Output Name columns in the Application Designer for Business Interlinks.
- A string containing the default value, if any.
- An attribute: ENUM_VAR_ATTR_REQUIRED if this data member is required, ENUM_VAR_ATTR_NONE if not required.

Parameters

None.

Returns

None.

Example

In the following method, the **Add** method adds the category names Transactions and All Transactions to the list of categories.

```
Private Function PsIoDriver_GetCategories(?
 ByVal eCatType As PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED,?
 ByVal plstCriterialNames As PSIODRIVERLib.IPsEnumString) As Long
   '
   Dim objPsBstr As PsBstr
   Set objPsBstr = plstCriterialNames.Add
   objPsBstr.Data = "Transactions"
   Set objPsBstr = plstCriterialNames.Add
   objPsBstr.Data = "All Transactions"
   PsIoDriver_GetCategories = True
   '
End Function
```

In the following method, the **Add** method adds a configuration parameter named Output File Name.

```
Private Function PsIoDriver_GetParameterList(ByVal plstParams As?
 PSIODRIVERLib.IPsEnumVarInfo) As Long
    '
    Dim objVarInfo As VarInfo
    Set objVarInfo = plstParams.Add
    objVarInfo.Name = "Output_File_Name"
```

```
    objVarInfo.DisplayName = ""
    objVarInfo.Value = ""
    objVarInfo.DataType = ""
    objVarInfo.Attribute = ENUM_VAR_ATTR_NONE
    PsIoDriver_GetParameterList = True
    '
End Function
```

In the following method, the **Add** method adds two transaction names: add numbers and concatenate strings.

```
Private Function PsIoDriver_GetTransactionByCategory(ByVal bstrFilter As String,?
 ByVal bstrCriteria As String, ByVal plstTransNames As PSIODRIVERLib.IPsEnumString)?
 As Long
    '
    Dim objPsTransactCategory As PsBstr
    Set objPsTransactCategory = plstTransNames.Add
    objPsTransactCategory.Data = "add numbers"
    Set objPsTransactCategory = plstTransNames.Add
    objPsTransactCategory.Data = "concatenate strings"
    PsIoDriver_GetTransactionByCategory = True
    '
End Function
```

In the following method, the **Add** method adds the input parameters of number_1 and number_2 and the output parameter of sum to the add numbers transaction.

```
Private Function PsIoDriver_GetTransactionByName(ByVal bstrTransName As String,?
 ByVal plstInput As PSIODRIVERLib.IPsEnumVarInfo,?
 ByVal plstOutput As PSIODRIVERLib.IPsEnumVarInfo) As Long
   '
   Dim objVarInfo As VarInfo

   If bstrTransName = m_bstrAddNumber Then
      Set objVarInfo = plstInput.Add
      objVarInfo.Name = "number_1"
      objVarInfo.DisplayName = ""
      objVarInfo.Value = ""
      objVarInfo.DataType = 0
      objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED

      Set objVarInfo = plstInput.Add
      objVarInfo.Name = "number_2"
      objVarInfo.DisplayName = ""
      objVarInfo.Value = ""
      objVarInfo.DataType = 0
      objVarInfo.Attribute = ENUM_VAR_ATTR_NONE

      Set objVarInfo = plstOutput.Add
      objVarInfo.Name = "sum"
      objVarInfo.DisplayName = ""
      objVarInfo.Value = ""
      objVarInfo.DataType = 0
      objVarInfo.Attribute = ENUM_VAR_ATTR_REQUIRED
```

```
        PsIoDriver_GetTransactionByName = True
    End If
End Function
```

AddInput

Syntax

C++:

```
BOOL AddInput(const VarInfo& var);
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

The class is TransactionDef.

The C++ **AddInput** method adds an input parameter to a transaction. The input parameter is of type VarInfo. VarInfo consists of the following structure:

- PSIOString name: the name of the parameter.
- PSIOString default: any default value for this parameter.
- DataType datatype: the data type of this parameter.
- BOOL required: indicates if the parameter is required.

Parameters

Parameter	Description
var	the variable that is added as an input parameter.

Returns

True if the input parameter was added to the transaction, False otherwise.

Value	Meaning
BOOL, Long	True if the input parameter was added to the transaction, False otherwise.

Example

In the following method, the **AddInput** method adds the input parameters number_1 and number_2 to the add numbers transaction.

```
BOOL SimpleDriver::GetTransactionByName(const PSIOString& strTransName,?
 TransactionDef& transDef)
{
   transDef.SetTransactionName(strTransName);
```

```
    if(strTransName == _T("add numbers"))
    {
       transDef.AddInput(VarInfo( _T("number_1"),?
  DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
  eVAR_ATTR_REQUIRED));
       transDef.AddInput(VarInfo( _T("number_2"),?
  DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
  eVAR_ATTR_NONE));
       transDef.AddOutput(VarInfo( _T("sum"),?
  DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
  eVAR_ATTR_REQUIRED));
       return TRUE;
    }
    else
       return FALSE;
}
```

AddDataMemberDef

Syntax

C++:

```
BOOL AddDataMemberDef(const VarInfo& var);
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

The class is ClassDef.

The C++ **AddDataMemberDef** method adds a data member to a class. The data member is of type VarInfo. VarInfo consists of the following structure:

- PSIOString name: the name of the data member.
- PSIOString default: any default value for this data member.
- DataType datatype: the data type of this data member.
- BOOL required: indicates if the data member is required.

Parameters

Parameter	Description
var	the variable that is added as a data member.

Returns

True if the data member was added to the class, False otherwise.

Value	Meaning
BOOL, Long	True if the data member was added to the class, False otherwise.

Example

In the following method, the **AddDataMemberDef** method adds the data members sender, cc, subject, date, and body to the Emessage class.

```
BOOL  EmailDriver::GetClassByName(const PSIOString& strClassName, ClassDef& classes)
{
   if(strClassName == "Emessage")
   {
      classes.SetClassName(strClassName);
      classes.AddDataMemberDef(VarInfo(_T("sender"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("cc"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("subject"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("date"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("body"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      return TRUE;
   }
   return FALSE;
}
```

AddOutput

Syntax

C++:

```
BOOL AddOutput(const VarInfo& var);
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

The class is TransactionDef.

The C++ **AddOutput** method adds an output parameter to a transaction. The output parameter is of type VarInfo. VarInfo consists of the following structure:

- PSIOString name: the name of the parameter.
- PSIOString default: any default value for this parameter.
- DataType datatype: the data type of this parameter.
- BOOL required: indicates if the parameter is required.

Parameters

Parameter	Description
var	the variable that is added as an input parameter.

Returns

True if the output parameter was added to the transaction, False otherwise.

Value	Meaning
BOOL, Long	True if the output parameter was added to the transaction, False otherwise.

Example

In the following method, the **AddOutput** method adds the output parameter sum to the add numbers transaction.

```
BOOL SimpleDriver::GetTransactionByName(const PSIOString& strTransName,?
 TransactionDef& transDef)
{
   transDef.SetTransactionName(strTransName);

   if(strTransName == _T("add numbers"))
   {
      transDef.AddInput(VarInfo( _T("number_1"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      transDef.AddInput(VarInfo( _T("number_2"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_NONE));
      transDef.AddOutput(VarInfo( _T("sum"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      return TRUE;
   }
   else
      return FALSE;
}
```

SetClassName

Syntax

C++:

```
BOOL SetClassName(const PSIOString& strClassName);
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

The class is ClassDef.

The C++ **SetClassName** method set the name of a class.

Parameters

Parameter	Description
StrClassName	the name of the class.

Returns

True if the data member was added to the class, False otherwise.

Value	Meaning
BOOL, Long	True if the data member was added to the class, False otherwise.

Example

In the following method, the **SetClassName** method sets the class name to be the PSIOString that is passed into the GetClassByName method.

```
BOOL  EmailDriver::GetClassByName(const PSIOString& strClassName, ClassDef& classes)
{
   if(strClassName == "Emessage")
   {
      classes.SetClassName(strClassName);
      classes.AddDataMemberDef(VarInfo(_T("sender"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("cc"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("subject"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("date"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
```

```
 eVAR_ATTR_REQUIRED));
      classes.AddDataMemberDef(VarInfo(_T("body"),?
 DataType(IOC_DATATYPE_STRING, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      return TRUE;
   }
   return FALSE;
}
```

SetTransactionName

Syntax

C++:

```
BOOL SetTransactionName(const PSIOString& strTransactionName);
```

Description

If there is no XML design-time plug-in for your runtime plug-in, you write the design-time functionality with the dynamic catalog methods. This is one of those methods. If you have an XML design-time plug-in, do not write this method.

The class is TransactionDef.

The C++ **SetTransactionName** method set the name of a transaction.

Parameters

Parameter	Description
StrTransactionName	the name of the transaction.

Returns

True if the input parameter was added to the transaction, False otherwise.

Value	Meaning
BOOL, Long	True if the input parameter was added to the transaction, False otherwise.

Example

In the following method, the **SetTransactionName** method set the transaction name to be the PSIOString that is passed into the GetTransactionByName method.

```
BOOL SimpleDriver::GetTransactionByName(const PSIOString& strTransName,?
 TransactionDef& transDef)
{
   transDef.SetTransactionName(strTransName);

   if(strTransName == _T("add numbers"))
```

```
   {
      transDef.AddInput(VarInfo( _T("number_1"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      transDef.AddInput(VarInfo( _T("number_2"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_NONE));
      transDef.AddOutput(VarInfo( _T("sum"),?
 DataType(IOC_DATATYPE_INT, _T("")), _T(""), _T(""), _T(""),?
 eVAR_ATTR_REQUIRED));
      return TRUE;
   }
   else
      return FALSE;
}
```

CHAPTER 18

Configuring PSINTERLINKS as a WebApp for Distributed Business Interlinks

This chapter provides an overview of PSINTERLINKS configuration as a WebApp for distributed Business Interlinks and discusses how to:

- Configure PSINTERLINKS as a WebApp on WebSphere.
- Configure PSINTERLINKS as a WebApp on WebLogic.
- Ensure that PSINTERLINKS is installed successfully and works for either WebLogic or WebSphere.

Understanding Configuration PSINTERLINKS as a WebApp for Distributed Business Interlinks

When loading a distributed Business Interlink Runtime plug-in (residing on a web server, as opposed to 2-tier or 3-tier, residing on an application server or a client), you must configure PSINTERLINKS as a WebApp on either WebSphere or WebLogic.

Configuring PSINTERLINKS as a WebApp on WebSphere

To configure Business Interlinks for WebLogic and WebSphere, update the PATH in C:\Apps\WebSphere\AppServer\bin\startServer.bat file. Add this to the PATH:

```
%WAS_HOME%/installedApps/peoplesoft/PSINTERLINKS/Web-inf/bin/winx86
```

The SET PATH then resembles:

```
SET PATH=%WAS_HOME%/installedApps/peoplesoft/PSINTERLINKS/Web-inf/bin/winx86;?
 %WAS_HOME%/bin;%JAVA_HOME%/bin;%JAVA_HOME%/jre/bin;%PATH%
```

You can backup the startServer.bat file into a PS-startServer.bat and then update the startServer.bat file with PATH changes.

Configuring PSINTERLINKS as a WebApp on WebLogic

To configure for WebLogic, you must set up the PATH in the setEnv.cmd file, and you must install your WebApp.

Setting Up the PATH in the setEnv.cmd file

The setEnv.cmd file is in $WEBLOGIC_HOME\config\peoplesoft, where $WEBLOGIC_HOME is the location of the WebLogic installation. This is commonly <PS_HOME>\webserv\peoplesoft \applications\peoplesoft\PSINTERLINKS\WEB_INF\bin\WINX86.

To set up the PATH in the setEnv.cmd file

1. Open the setEnv.cmd file in an editor, and find the remark section at the end of the file.

```
@rem To enable the PSINTERLINKS Web Application, define the
@rem PSINTERLINKS
@rem web application in the WebLogic administrative console, append
@rem the following entry to your PATH below, and restart WebLogic
@rem Server.
@rem ;.\config\%DOMAIN_NAME%\applications\PSINTERLINKS\WEB-INF\bin\winx86
@rem
```

2. Find the PATH variable in the setEnv.cmd file, and append the following onto it:

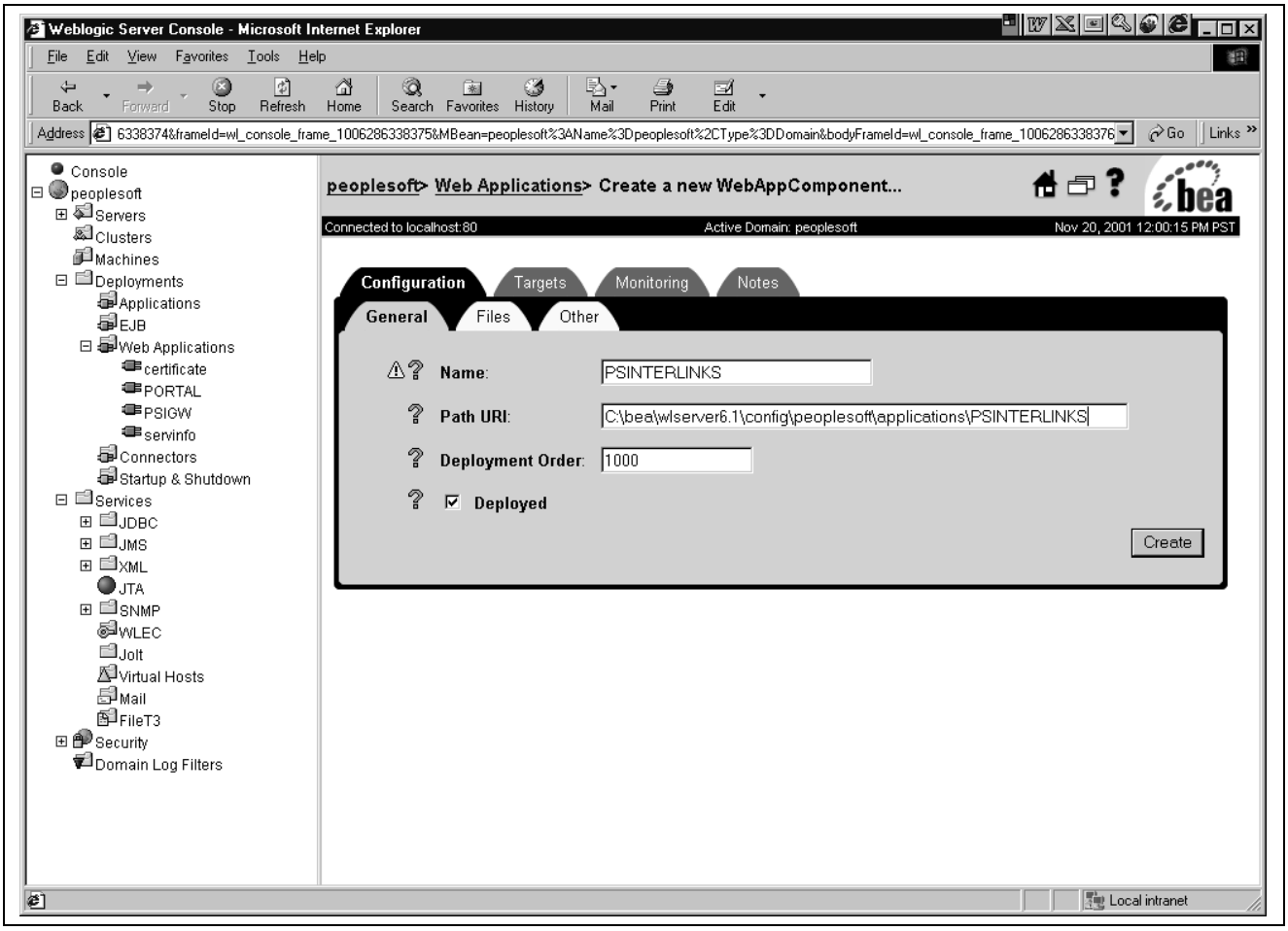
```
;.\config\%DOMAIN_NAME%\applications\PSINTERLINKS\WEB-INF\bin\winx86
```

   The PATH then resembles:

```
PATH=%WL_HOME%\bin;%JAVA_HOME%\bin;%PATH%;?
 .\config\%DOMAIN_NAME%\applications\PSINTERLINKS\WEB-INF\bin\winx86
```

Installing your WebApp

You must also install your WebApp.



WebLogic Server Console

To install your WebApp:

1. Access the WebLogic Server administrative console at http://<webserver>/console, where <webserver> is the location of the WebLogic server.
2. When prompted for a user name and password, specify the WebLogic system ID and password.

   In the default WebLogic Server install, the ID and password are ’system’ and ’password’. If the default was not used, specify the password supplied during the WebLogic Server installation.
3. Select Peoplesoft, Deployments, Web Applications, and then click Configure a Web Application.
4. On the Configuration, General tabs, fill out the following:

   Name: PSINTERLINKS

   Path URL: <PS_HOME>\webserv\peoplesoft\.
5. Click the Targets tab.
6. Select the PIA server in the Available box, and click the right arrow to move it to the Chosen box.
7. Click Apply.

Ensuring that PSINTERLINKS is installed successfully and works for either WebLogic or WebSphere

To ensure that PSINTERLINKS is installed successfully and works for either WebLogic or WebSphere

1. Cut and paste the HTML section (as below) into a file named test.html:

```
<html>
<head>
<title>Test BusInterlinkServLet</title>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<center>
<br>
<blink><h1><b><i><font color="ff00ff">Test BusInterlinkServLet</font></i></b></h1>?
 </blink>
<br>
</center>
<h1><b>Set a Home directory for all your plug-in's</h1><b>
<form method=get action="http://your_web_server/PSINTERLINKS/BusInterlinkServLet">
<input type="text" size=30 name=homedir value="">
<br>
<br>
<input type="submit" value="Ok">
<input type="reset" value="Clear">
</form>
<h1><b>Test PsioloaderJ</h1><b>
<form method=post action="http://your_web_server/PSINTERLINKS/BusInterlinkServLet">
<br>
<br>
<input type="text" size=30 name=xml value="test">
<input type="submit" value="Ok">
</form>
</body>
</html>
```

2. Modify *your_web_server* to be your web server name and port number, and save the test.html file.
3. Double click on test.html.
4. In the first Text box, put in the path containing the Business Interlink supporting libraries.

   On WebSphere it is:

   ```
   C:\APPS\WebSphere\AppServer\installedApps\peoplesoft\PSINTERLINKS\Web-inf\bin\winx86
   ```

   On WebLogic it is (where c:\bea\wlserver6.1 is $WEBLOGIC_HOME, the location of the WebLogic installation):

   ```
   <PS_HOME>\webserv\peoplesoft\applications\peoplesoft\PSINTERLINKS\WEB_INF\bin\WINX86
   ```

5. Click the first OK button.

   The first text box and OK button tests the HTTP *get* functionality of the servlet BusInterlinkServLet. If it works, it means that the BusInterlinkServLet servlet is installed correctly.

6. Click the second OK button.

The second text box and OK button tests the HTTP *post* functionality of the servlet BusInterlinkServLet. If it works, it means the post functionality works. It also loads the BI supporting libraries successfully.

PART 4

Business Interlinks for XML

CHAPTER 19

Writing an XML Design-time Plug-in Using the pshttpenable Runtime Plug-in

This chapter provides an overview of XML design-time plug-ins that use the pshttpenable runtime plug-in, and discusses how to:

- Configure parameters for the pshttpenable runtime plug-in.
- Write the pshttpenable runtime plug-in input, output, and class parameters.
- Escape XML restricted characters.
- Use safe characters.
- Use an example of the XML design-time plug-in that uses pshttpenable.
- Read an XML string: pshttpenable runtime plug-in.
- Use the XML template for the pshttpenable runtime plug-in.

Understanding the pshttpenable Runtime Plug-in

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

The pshttpeneable runtime plug-in is a self-describing program that encapsulates such external systems. It executes HTTP functions GET and POST for sending XML requests to the external system, and sets the data document for the response data. You can refer to the URL below for more detail about the GET and POST HTTP functions.

See http://www.w3.org/Protocols/rfc2616/rfc2616.html

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

Note. The generic plug-in (pshttpenable) must reside where PeopleTools is installed.

Configuring Parameters for the pshttpenable Runtime Plug-in

This section provides an overview of the configuration parameters for the pshttpenable runtime plug-in and discusses how to:

- Specify Accept Type Headers: Accept_Type.
- Specify Content Type Headers: Content_Type.
- Specifying a Cookie: Cookie.
- Add HTTP Headers: HttpHeader.
- Specify Input Data Type: InDataType.
- Debug Input Data: Input_File_Name.
- Specify The Merchant URL: MerchantURL.
- Create a DOCTYPE Statement: Metatag.
- Specify the Method Parameters: Method.
- Specify Output Data Type: OutDataType.
- Specify User Password: PASSWORD.
- Set Proxy Authentication: Proxy-Authentication.
- Set Proxy Authorization: Proxy-Authorization.
- Debug Request Data: Request_File_Name.
- Debug Request Data: Response_File_Name.
- Debug Response Data: Simulated_Response_File_Name.
- Specify Secure Link: SSL.
- Specify the pshttpenable runtime plug-in: <URL>.
- Specify User ID: USERID.
- Check for Errors in XML Syntax: XML_Validation.
- Allow or not allow redirects: Redirect.

Understanding the Configuration Parameters for the pshttpenable

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

See Chapter 8, “Writing an XML Design-time Plug-in,” Writing The Tags in an XML Design-time Plug-in, page 106 and Chapter 7, “Designing Business Interlink Transactions and Classes,” Listing Configuration Parameters, page 95.

Specifying Accept Type Headers: Accept_Type

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"/>
```

Specifying Content Type Headers: Content_Type

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" />
```

Specifying a Cookie: Cookie

The Cookie parameter allows you to use HTTP cookies.

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

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

Within PeopleCode, you must create a global string to store the cookie information. Before you call the Business Interlink Execute method, set the Cookie parameter. After the Execute function, retrieve the Cookie back into the global string, using the Business Interlink property Cookie. The Generic plug-in, pshttpenable, does not keep the cookie for you.

```
Global String  &cookie;

&MyBi = GetInterlink(Interlink.MYBI);

&MyBi.Cookie = &cookie;
&Ret = &MyBi.Execute();
&cookie = &MyBi.Cookie;
```

Adding HTTP Headers: HttpHeader

The HttpHeader parameter allows you to enter any HTTP headers, including custom headers. The word HttpHeader must be followed by a colon with no spaces in between. The old header settings (provided only for **Accept** and **Content Type**) are valid and supercede those set in the new way as long as the header names match (Content-Type and Accept, respectively). For Content-Length, that post is determined and set at runtime by the pshttpenable runtime plug-in.

Here are examples of how you could write the <*parameter name = "HttpHeader"*> tag.

```
<parameter name="HttpHeader:Accept" type="enum(text/xml/html, text/xml, text/html,?
 */*)" required="true" default="text/xml/html"/>

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

<parameter name="HttpHeader:Accept-Language" type="enum(en,da)"?
 required="true" default="text/xml/html"/>
```

Specifying Input Data Type: InDataType

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.

Debugging Input Data: Input_File_Name

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.

Specifying The Merchant URL: MerchantURL

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.

When setting the URL for the merchant API, you might have to enter a userID, password, and machine name to access the merchantAPI. Within the MerchantURL parameter, you can set the merchant URL as follows:

```
http://<userID>:<password>@<machine>/path_on_web_server
```

For example, you might be trying to do a request to return an inbox in HTML, and have it logon the user also. If the machine is named mach1 with the userID of user1 and password of passwrd1, you could set the URL for the merchantAPI as follows:

```
http://user1:passwrd1@mach1/exchange/logonfrm.asp?mailbox=user1
```

Create a DOCTYPE Statement: Metatag

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> " />
```

Specifying the Method Parameters: Method

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.

Specifying Output Data Type: OutDataType

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.

Specifying User Password: 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.

Setting Proxy Authentication: Proxy-Authentication

The Proxy-Authentication parameter allows you to set proxy authentication.

Write the <*parameter name = "Proxy-Authentication"*> tag as follows.

```
<parameter name="Proxy-Authentication"
   type="string"
```

```
   default="userid:password"/>
```

where *userid* is the User ID, and *password* is the password.

Setting Proxy Authorization: Proxy-Authorization

The Proxy-Authorization parameter allows you to set proxy authorization.

Write the <*parameter name = "Proxy-Authorization"*> tag as follows.

```
<parameter name="Proxy-Authorization"
   type="string"
   default="authorrequirements"/>
```

where *authorrequirements* are the authorization requirements.

Debugging Request Data: Request_File_Name

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.

Debugging Request Data: Response_File_Name

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.

Debugging Response Data: Simulated_Response_File_Name

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.

Specifying Secure Link: SSL

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.

Specifying the pshttpenable Runtime Plug-in: <URL>

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

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

Specifying User ID: USERID

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.

Checking for Errors in XML Syntax: XML_Validation

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.

Allowing or not Allowing Redirects: Redirect

The optional Redirect parameter specifies if redirection is allowed at the merchant URL. If set to YES, or not specified, then redirects are allowed at the merchant URL. When a redirect occurs, return_status_message is set to 200. If Redirect is set to NO, redirects are not allowed; however, if a redirect is attempted , return_status_message is set to 302.

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

```
<parameter name="Redirect"
   type="enum(YES,NO)"
   required="false"
   default="value"/>where value is either YES or NO
```

Writing the pshttpenable Runtime Plug-in Input, Output, and Class Parameters

You write the input, output, and class parameters to match the inputs, outputs, and classes for the transaction that is provided by the merchant.

See Chapter 7, “Designing Business Interlink Transactions and Classes,” page 93.

Escaping XML Restricted Characters

Anywhere you pass in text or data in the XML design-time plug-in, including the configuration parameters, all XML restricted characters (such as < and >) need to be escaped.

- < must be %60
- > must be %62
- " must be %34

Using Safe Characters

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

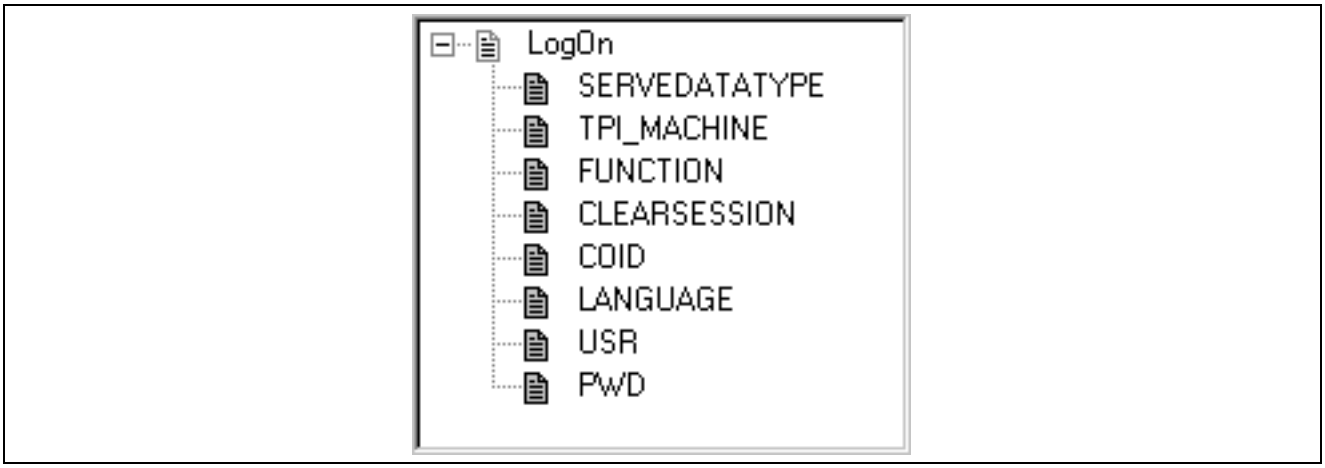
- 0-9
- A-Z
- a-z
- ? & (blank space)

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

Example of an XML Design-time Plug-in That Uses pshttpenable

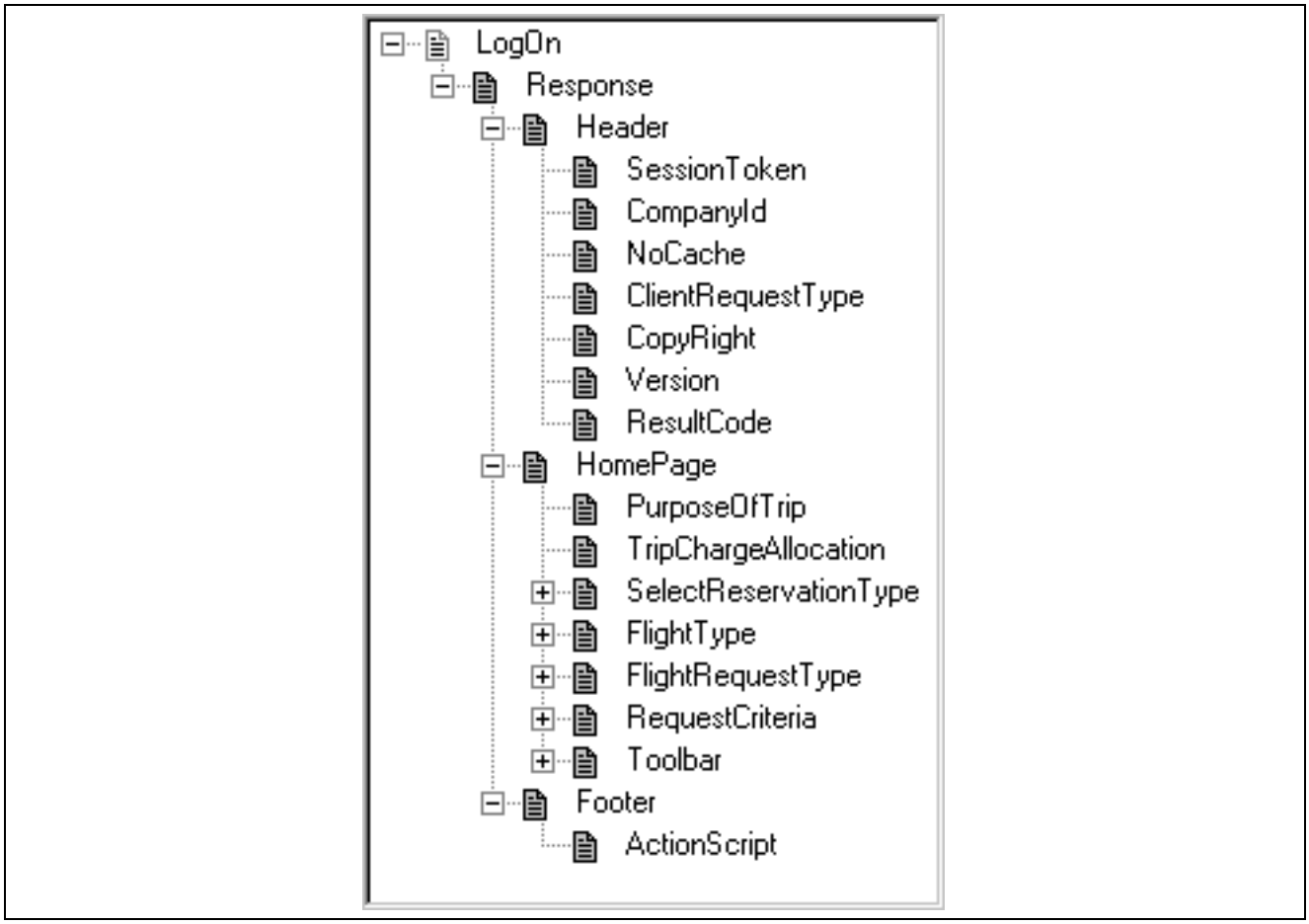
This example shows an example of an XML design-time plug-in for the Sabre LogOn transaction resulting from the XML code in this chapter. This plug-in uses the GET method.

Following is the input structure.



Sabre LogOn transaction inputs

Following is the output structure.



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="PEOPLESOFT"
               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>
```

Reading an XML String For the pshttpenable Runtime Plug-in

This section discusses how to:

- Read an XML string using Business Interlink plug-ins.
- View an example of a design-time plug-in for reading an XML String.
- View an example of PeopleCode for reading an XML string.

Reading an XML String Using Business Interlink Plug-ins

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.

To read an XML string using the pshttpenable runtime plug-in:

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.

Example of a Design-time Plug-in for Reading an 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"/>
```

```
      <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 in the example of a design-time plug-in that reads an XML string.

```
Local string &importXMLString;
Local Interlink &ImportXML;
Local BIDocs &InputDoc, &OutputDoc;
Local number &EXECRSLT;
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></errortext></error>?
<candidates><user><userid>9719</userid><username>KWoolf</username>?
<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></user>?
<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></user></candidates></postreqresponse>";

/* XMLString is the name of the single input. */
&ret = &InputDoc.AddValue("XMLString", &importXMLString);

&EXECRSLT = &ImportXML.Execute();
/* The Business Interlink document is now filled with the
      /* data from the input string. */
If (&EXECRSLT <> 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("postreqresponse.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;
```

Using the 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 20

Creating an Inbound Business Interlink

This chapter provides an overview about inbound Business Interlinks and discusses how to:

- Set up the configuration properties for inbound Business Interlinks.
- Set up the inbound Business Interlink environment for an IScript function.
- Test the inbound Business Interlink setup.
- Write the IScript function.
- View an example of an XML design-time plug-in used with an inbound Business Interlink.
- Use Apache or WebLogic web servers with inbound Business Interlinks.
- Use inbound Business Interlink methods.
- Use inbound Business Interlink properties.
- Use inbound Business Interlink functions.
- Use iScript methods.

Understanding Inbound Business Interlinks

This chapter lays out the steps you follow to create an inbound Business Interlink. An inbound Business Interlink allows an IScript function (PeopleCode program) to receive an XML request via HTTP from a third party system, and send an XML response to the third party system.

Prerequisities

If you want to send your XML request to PeopleSoft and receive XML responses, your transactions must be:

- Sessionless.
- Synchronized.
- Over the internet (HTTP).

Setting Up the Configuration Properties for Inbound Business Interlinks

In order to use inbound Business Interlinks, you must set up configuration properties in the PeopleTools Web Profile.

1. From your browser, select PeopleTools, Web Profile and click the Security tab.
2. In the XML Link group box, enter the user ID and password combination.

Check the XML Link Use Http For Same Server check box if you want to use the http protocol instead of https for requests issued by the XmlLink for content hosted on the same server as the XmlLink servlet. In cases such as this, the request is hosted on the same server, so the http protocol is preferred.

See *Enterprise PeopleTools 8.45 PeopleBook: Internet Technology*.

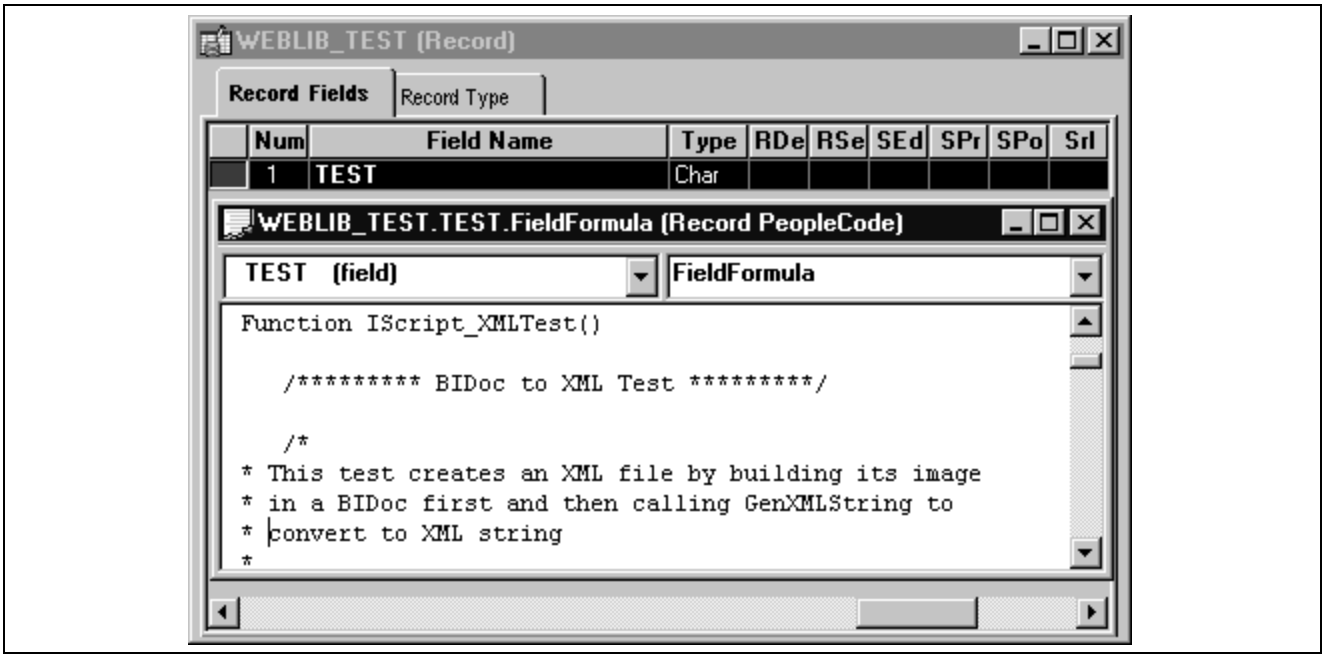
Setting Up the Inbound Business Interlink Environment for an IScript Function

This section discusses how to:

- Create the location for your IScript function (PeopleCode program).
- Register the IScript function by service name.
- Enter the service name into the XML Link Function Registry.

Creating the Location for the IScript Function (PeopleCode Program)

This section discusses how to create the location for your IScript function (PeopleCode program).



IScript function in Test Field: Field Formula

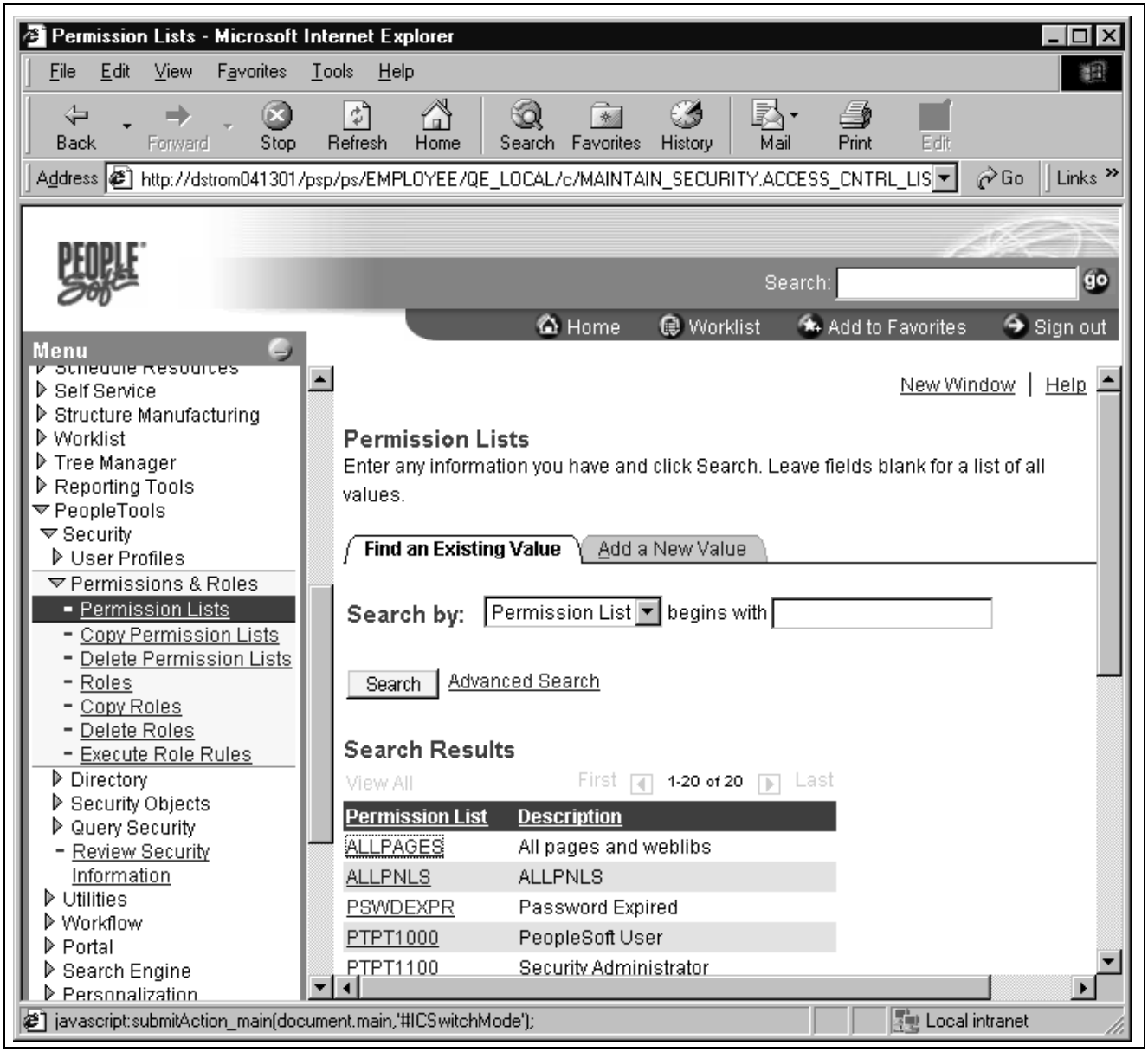
To create the location for your IScript function (PeopleCode program):

1. Open or create a record named WEBLIB_*name*, where *name* is the name that you want for your IScript record.
2. Within that record, select the FieldFormula field (FFo). Write your IScript function (PeopleCode program) within this field. The following shows the IScript function named IScript_Test, contained in the WEBLIB_test record, its TEST field containing the IScript function within FieldFormula.

See Chapter 20, “Creating an Inbound Business Interlink,” Writing the IScript Function, page 332.

Registering the IScript Function by Service Name

This section discusses how to register the IScript function by service name.

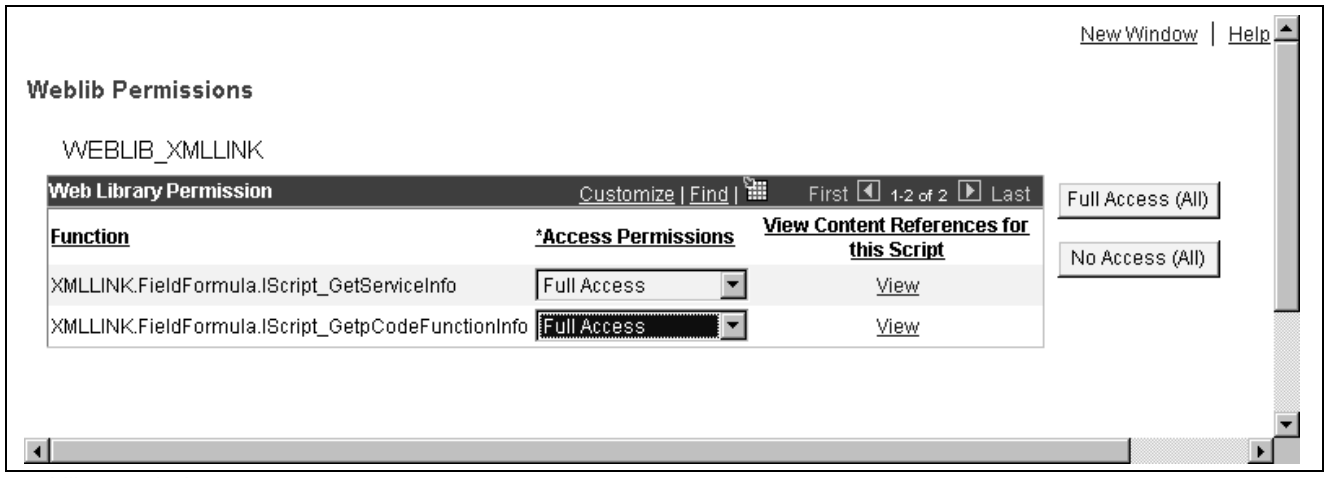


Permission Lists page

To register the IScript function by service name:

1. From your browser, select PeopleTools, Security, Permissions and Roles, Permission Lists.
2. Within the Permission Lists page, select the type of permission that you want from the Permission List listbox. For example, ALLPAGES provides the most generic permissions.
3. Within the Permission Lists page for the type of permissions you selected, click on the Web Libraries tab to display the web libraries page.
4. Associate a service name to the IScript function.
    a. Click on the Edit button next to your WEBLIB_*name*. If your WEBLIB_*name* does not appear in this panel, add it by clicking on one of the Add buttons.
    b. Within the WebLib Permissions page, select Full Access (All) for your IScript and click OK.

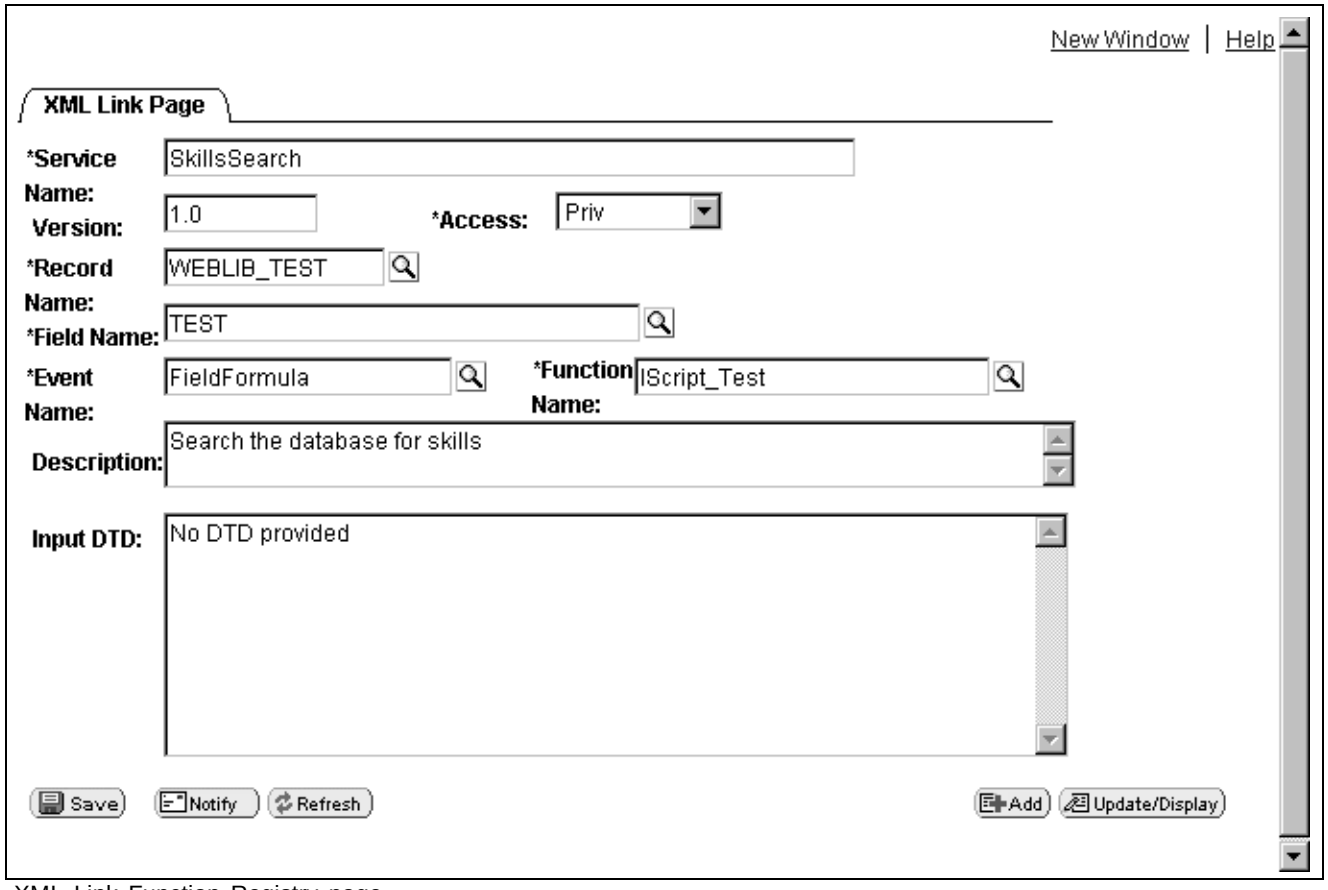
c. Grant security/permission to your WEBLIB_*name* by repeating the steps to associate a service name for WEBLIB_XMLLINK.



Weblib Permissions page

Enter the Service Name into the XML Link Function Registry

This section discusses how to enter the service name into the XML Link Function Registry.



XML Link Function Registry page

To enter the service name into the XML Link Function Registry:

1. From your browser, select PeopleTools, Utilities, Administration, XML Link Function Registry.

2. Within the Utilities − Use − XML Link Function Registry page, enter the name you want for your service. Then, in the following order, select from the drop-down lists, and then select Save:
    a. Record: Enter the name of the record containing your IScript function (PeopleCode program).
    b. Field: Enter the name of the record field.
    c. Event: Select Field Formula.
    d. Function Name: Select your IScript function name.

Testing the Inbound Business Interlink Setup

Access the xmllink servlet and check if your service appears in the list. Using the browser, get the URL http://*host*/servlet/xmllink/*PIApath*, where *host* is your server name, and *PIApath* is the default path for PIA (this is usually peoplesoft8). This will provide the list of registered services.

Test your set-up. For example, you can do a POST to the following URL:

```
http://host/servlet/xmllink/PIApath/servicename?userid=PTDMO&pwd=PTDMO)
```

where *host* is your server name, *PIApath* is the default path for PIA (this is usually peoplesoft8), and PTDMO is the username, and the password. *servicename* is the name of the service: in this example, that is SkillsSearch.

You can create a Business Interlink pshttpenable design-time plug-in to send an XML request and receive the XML response.

See Chapter 20, "Creating an Inbound Business Interlink," Viewing an Example of an XML Design-time Plug-in Used with an Inbound Business Interlink, page 337.

Writing the IScript Function

This section provides an overview of the IScript function tasks and discusses how to:

- Pass in a GET action parameter when accessing an XMLLink service.
- View an example of a PeopleCode Program for an inbound Business Interlink.
- View a listing of the XML request.
- View a listing of the XML response.

Understanding the IScript Function Tasks

The IScript function that you write performs the following tasks:

- Navigates the request XML using either the new BiDoc methods (listed at below) or populating data on an existing Business Interlink BiDoc using the fromXML(string) function.
- Does back-end processing to get data for the response.
- Creates the response XML using the new BiDocs methods (or populate an existing Business Interlink with addValue(string)).

Passing in a GET Action Parameter When Accessing an XMLLink Service

You can pass parameters in a GET when you access an XMLLink service by using name/value pairs and &runservice=yes in the URL.

On the browser or in some other HTTP GET utililty (BI Tester etc) set the following URL:

```
http://machinename:port#/xmllink/piapath/servicename/?userid=XXX&pwd=XXX?
&runservice=yes&abc=xyz&def=bbb
```

where &abc=xyz&def=bbb are any name-value parameter pairs you want to pass.

In the service IScript, use PeopleCode to retrieve the parameters. For example, this function retrieves and resends the values for the abc and def parameters.

```
Function IScript_Service()

   &blob = %Request.GetContentBody();
   &value = %Request.GetParameter("abc");
   %Response.Write(&value);
  &value = %Request.GetParameter("def");
   %Response.Write(&value);

End-Function;
```

Viewing an Example of a PeopleCode Program for an Inbound Business Interlink

Here is a PeopleCode program that performs these tasks for the inbound Business Interlink example in this chapter.

```
/********* Locals ***************************************/
Local BIDocs &rootDoc, &rootInDoc;
Local BIDocs &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local BIDocs &candidates;
Local BIDocs &user, &conid, &username, &comments, &role, &availability,?
 &rate, &relevancy, &hobbies;
Local BIDocs &location, &city, &state, &zip, &country;
Local BIDocs &desiredloc;
Local string &xmlString;
Local SQL &GetCandidatesSQL;


Function IScript_Test()

   /********* BIDoc to XML Test *********/

   /*
* This test creates an XML file by building its image
* in a BIDoc and then calling GenXMLString to convert to XML string
*
```

```
*/
   /************* Main *************/
   /* Receive the request object and convert to an XML string */
   &blob = %Request.GetContentBody();

   /* process the inbound xml(request)- Create a BiDoc */
   &rootInDoc = GetBiDoc(&blob);
   /* Use BiDocs methods to extract the skills value from this XML request. */
   &postreqDoc = &rootInDoc.GetDoc("postreq");
   &ret = &postreqDoc.GetValue("skills", &skills);


   /*create SQL object to query the Database*/
   &GetCandidatesSQL = CreateSQL("SELECT CONID, SKILLS, USERNAME, COMMENTS,
 SCENERY, DENSITY, BLANK, CITY, STATE, ZIP, COUNTRY, DESIREDLOC, ROLE,
 AVAILABILITY, RATE, RELEVANCY, RATING, HOBBIES FROM PS_CANDIDATES
 WHERE SKILLS =:1 ORDER BY CONID", &skills);



   /* Create the BiDoc which will have the same structure as the XML response*/
   &rootDoc = GetBiDoc("");

   /* add a processing instruction (the first line in the XML response) */
   &ret = &rootDoc.addProcessInstruction("<?xml version=""1.0""?>");

   /* create the postreqresponse tag for the XML response */
   &postreqresponse = &rootDoc.createElement("postreqresponse");

   /* create the error tag within the postreqresponse tag */
   &error = &postreqresponse.createElement("error");

   /* create the errorcode and errortext tags within the error tag */
   &errorcode = &error.createElement("errorcode");
  &ret = &errorcode.addText("1");
   &errortext = &error.createElement("errortext");

/* create the candidates tag within the postreqresponse tag */
   &candidates = &postreqresponse.createElement("candidates");

   /* Fetch all the rows for users and create and populate the /*
   /* candidates tag with user tags and their data dynamically */
   While (&GetCandidatesSQL.Fetch(&conidvalue, &skillsvalue, &usernamevalue,?
 &commentsvalue, &sceneryvalue, &densityvalue, &blankvalue, &cityvalue,?
 &statevalue, &zipvalue, &countryvalue, &desiredlocvalue, &rolevalue,?
 &availabilityvalue, &ratevalue, &relevancyvalue, &ratingvalue, &hobbiesvalue))
      &user = &candidates.createElement("user");
      /* create the tags within the user tag to contain data */
      &conid = &user.createElement("conid");
      &ret = &conid.addText(&conidvalue);
```

```
      &username = &user.createElement("username");
      &ret = &username.addText(&usernamevalue);
      &comments = &user.createElement("comments");
      &ret = &comments.addText(&commentsvalue);
      &location = &user.createElement("location");


      /* Add some attributes to the location tag */
      &ret = &location.addAttribute("scenery", &sceneryvalue);
      &ret = &location.addAttribute("density", &densityvalue);
      &ret = &location.addAttribute("blank", &blankvalue);

      /* Add tags and their data to the location tag */
      &city = &location.createElement("city");
      &ret = &city.addText(&cityvalue);
      &state = &location.createElement("state");
      &ret = &state.addText(&statevalue);
      &zip = &location.createElement("zip");
      &ret = &zip.addText(&zipvalue);
      &country = &location.createElement("country");
      &ret = &country.addText(&countryvalue);
      &desiredloc = &user.createElement("desiredloc");
      &ret = &desiredloc.addText(&desiredlocvalue);
      &role = &user.createElement("role");
      &ret = &role.addText(&rolevalue);
      &availability = &user.createElement("availability");
      &ret = &availability.addText(&availabilityvalue);
      &rate = &user.createElement("rate");
      &ret = &rate.addText(&ratevalue);
      &relevancy = &user.createElement("relevancy");
      &ret = &relevancy.addText(&relevancyvalue);
      &rating = &user.createElement("rating");
      &ret = &rating.addText(&ratingvalue);

      /* Add a comment */
      &ret = &user.addComment("Added a new tag for hobbies");
      /* Add another tag, hobbies, to the user tag */
      &hobbies = &user.createElement("hobbies");
      &ret = &hobbies.addText(&hobbiesvalue);
   End-While;


   &GetCandidatesSQL.Close();

   /* convert the BiDoc to an XML string */
   &xmlString = &rootDoc.GenXMLString();

/* Send the Response Object */
   %Response.Write(&xmlString);
```

```
End-Function;
```

Viewing a Listing of the XML Request

The PeopleCode program written for the inbound Business Interlink example expects to receive the following XML request:

```
<?xml version="1.0"?>
<postreq>
   <userid>14682</userid>
   <password>play2win</password>
   <email>joe_blow@peoplesoft.com</email>
   <projtitle></projtitle>
   <projref>PSFT</projref>
   <location>35</location>
   <country>USA</country>
   <startdate>12-Dec-1999</startdate>
   <duration>lessthan1month</duration>
   <mincomp>45</mincomp>
   <compunit>Hourly</compunit>
   <compcurrency>$</compcurrency>
   <role>25</role>
   <expireday>30</expireday>
   <description>testing...</description>
   <skills>JAVA</skills>
</postreq>
```

Viewing a Listing of the XML Response

The PeopleCode program written for the inbound Business Interlink example will send the following XML request, assuming that the database contains Joe Blow and Jane Doe, and they both know Java:

```
<?xml version="1.0"?>
<postreqresponse>
   <error>
      <errorcode>1</errorcode>
      <errortext></errortext>
   </error>
   <candidates>
      <user>
         <conid>0000001000</conid>
         <username>Joe Blow</username>
         <comments>Joe Blow comment: Goodbye!</comments>
         <location scenery="great" density="low" blank="eh?">
            <city>San Rafael</city>
            <state>CA</state>
            <zip>94522</zip>
            <country>US</country>
         </location>
         <desiredloc>No Preference</desiredloc>
```

```
            <role>Database</role>
            <availability>2000-10-28</availability>
            <rate>0000000045</rate>
            <relevancy>10</relevancy>
            <rating>0</rating>
            <!--Added a new tag for hobbies-->
            <hobbies>saying goodbye</hobbies>
         </user>
         <user>
            <conid>0000001001</conid>
            <username>Jane Doe</username>
            <comments>Jane comment: hello!</comments>
            <location scenery="so-so" density="dense" blank="some stuff">
               <city>Concord</city>
               <state>NJ</state>
               <zip>32444</zip>
               <country>US</country>
            </location>
            <desiredloc>South Bay</desiredloc>
            <role>Developer</role>
            <availability>2000-10-31</availability>
            <rate>0000000100</rate>
            <relevancy>19</relevancy>
            <rating>2</rating>
            <!--Added a new tag for hobbies-->
            <hobbies>saying hello</hobbies>
         </user>
      </candidates>
   </postreqresponse>
```

Viewing an Example of an XML Design-time Plug-in Used with an Inbound Business Interlink

The following XML code is a Business Interlink XML design-time plug-in. This plug-in allows a PeopleCode application to send the XML request that is used as the inbound Business Interlink example, and then receive the XML response that is used by the inbound Business Interlink example.

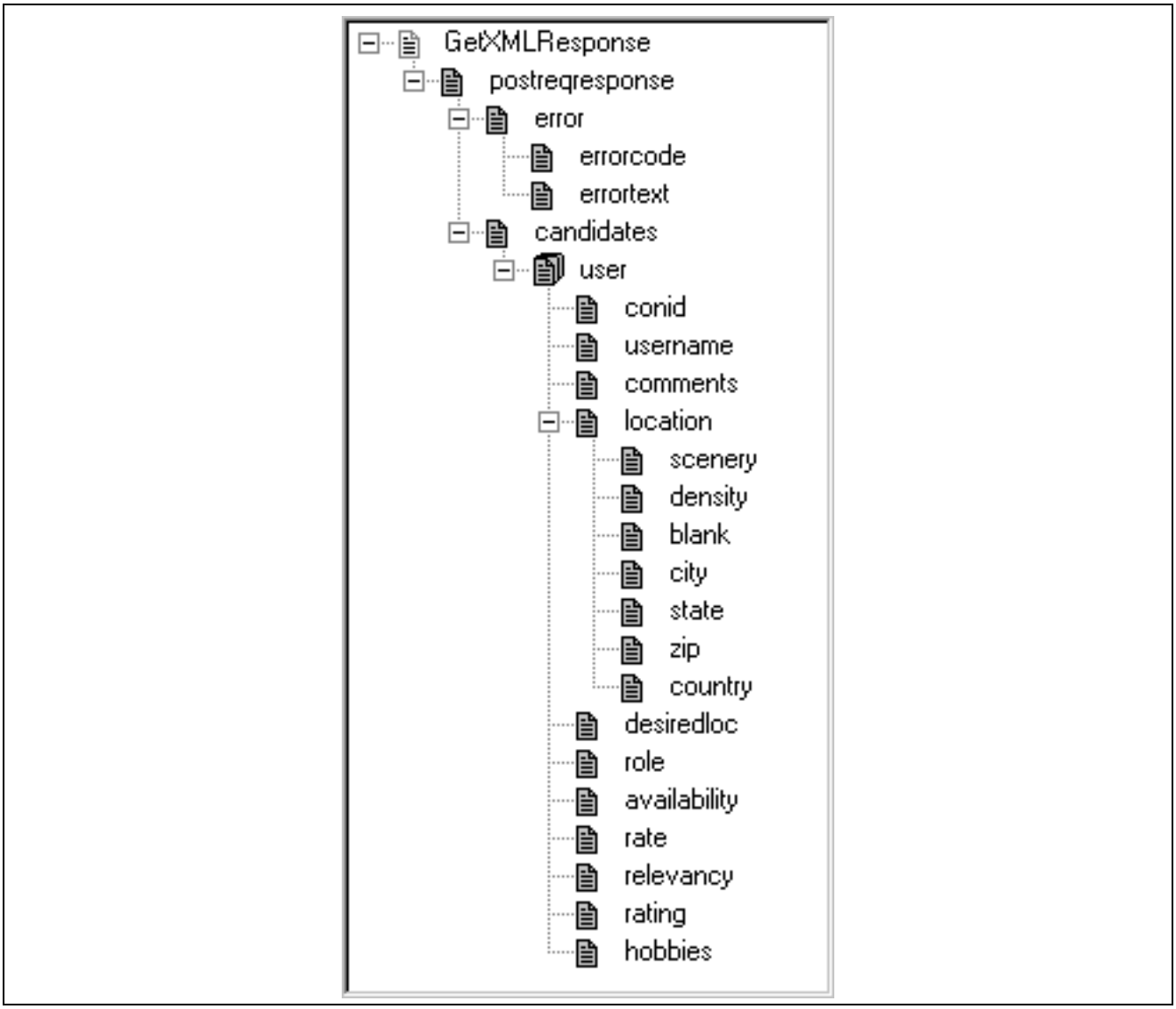
This plug-in consists of a transaction whose inputs are the XML request, and whose outputs are the XML response.

See Chapter 19, “Writing an XML Design-time Plug-in Using the pshttpenable Runtime Plug-in,” page 307.

For reference, here is the structure of the inputs for this XML design-time plug-in.

Transaction inputs: XML request

For reference, here is the structure of the outputs for this XML design-time plug-in.



Transaction outputs: XML response

```
<?xml version="1.0"?>
<interface_driver>
   <general_info>
      <description>Inbound Business Interlink Demo Prototype Plug-in</description>
      <version>1</version>
      <lastupdate>04/19/00</lastupdate>
      <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="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="http://localhost/servlets/xmllink/peoplesoft8/SkillsVillageSearch"/>
      <parameter name="InDataType" type="enum(XML,HTML,DHTML)"
         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="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=""/>
      <parameter name="userid" type="string"  default="PTDMO"
         required="true"/>
      <parameter name="pwd" type="string"  default="PTDMO"
         required="true"/>
   </config_parameters>

   <class_catalog>
      <category name="Inbound Business Interlink Demo">
<!-- The postreq class contains the structure for the input for this transaction. -->
         <class name="postreq">
            <member name="userid" type="string" default="14682"
               required="false"/>
            <member name="password" type="string" default="play2win"
               required="false"/>
            <member name="email" type="string"
               default="joe_blow@peoplesoft.com"
               required="false"/>
            <member name="projtitle" type="string" default=""
```

```
                required="false"/>
             <member name="projref" type="string" default="PSFT"
                required="false"/>
             <member name="location" type="int" default="35"
                required="false"/>
             <member name="country" type="string" default="USA"
                required="false"/>
             <member name="startdate" type="string"
                default="12-Dec-1999" required="false"/>
             <member name="duration"
                type="enum(lessthan1month, 1to3months, 3to6months,
                6to12months, 12plusmonths)"
                default="lessthan1month" required="false"/>
             <member name="mincomp" type="float" default="45"
                required="false"/>
             <member name="compunit"
                type="enum =(Daily, Hourly, Monthly, Weekly,Yearly)"
                default="Hourly" required="false"/>
             <member name="compcurrency" type="enum =($,?)"
                default="$" required="false"/>
             <member name="role" type="int" default="25"
                required="false"/>
             <member name="expireday" type="enum(30,60,90)"
                default="30" required="false"/>
             <member name="description" type="string"
                default="testing..." required="false"/>
             <member name="skills" type="string" default="JAVA"
                required="false"/>
          </class>

<!--The postreqresponse class contains the structure for the output for this?
 transaction.-->
! The postrequestresponse class contains the error class and the
! candidates class.
          <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="userinfo" default=""/>
          </class>
```

```
            <class name="userinfo">
               <member name="conid" 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="location" 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"/>
               <member name="hobbies" type="string" default=""
                  required="false"/>
            </class>
            <class name="location">
               <member name="scenery" type="string"
                  attribute="xml_attr" default="" required="false"/>
               <member name="density" type="string"
                  attribute="xml_attr" default="" required="false"/>
               <member name="blank" type="string" attribute="xml_attr"
                  default="" required="false"/>
               <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>

   !Here is the transaction containing the input/request postreq class,
   !and the output/response postreqresponse class.
      <trans_catalog>
         <category name="Inbound Business Interlink Demo">
            <transaction name="GetXMLResponse">
```

```
            <input_list>
               <input name="postreq" type="object"
                  classname="postreq" default="" required="false"/>
            </input_list>
            <output_list>
               <output name="postreqresponse" type="object"
                  classname="postreqresponse" required="true"/>
            </output_list>
         </transaction>
      </category>
   </trans_catalog>
</interface_driver>
```

The postreqresponse class contains the structure for the output for this transaction. This chapter contains an example of the XML response that this produces for the inbound Business Interlink. The postreq class contains the structure for the input for this transaction. This chapter also contains an example of the XML request that this produces for the inbound Business Interlink.

See Chapter 20, “Creating an Inbound Business Interlink,” Viewing a Listing of the XML Response, page 336.

Using Apache or WebLogic Web Servers with Inbound Business Interlinks

If you use Apache or WebLogic web servers and SSL, there will be an error when the servlet tries to connect to the iScript. To avoid this problem, post directly to the iScript, bypassing the servlet.

Using the Inbound Business Interlink Methods and Properties

The inbound Business Interlink methods allow you to parse an XML request and build an XML response.

The GetContentBody Request object method converts the request content (an XML request) into an XML string. You can then use this string with the **GetBiDoc** function to create a BiDoc structure that is the same shape and contains the same data as the XML string.

Once you have the BiDocs structure containing the XML request, you can:

- Use the **GetNode** method to get one of the XML elements.
- Use the **NodeType** method to get the type of the node (element, processing instruction, comment).
- Use the **NodeName** property to get the name of the element.
- Use the **NodeValue** property to get the value of the element.

You can also use the standard BiDocs methods (such as GetDoc, GetValue) to retrieve information from this BiDocs object.

When an XML element contains attributes, the **AttributeCount** property gets the number of attributes, the **GetAttributeName** method gets the name of an attribute, and the **GetAttributeValue** method gets the value of an attribute.

To build an XML response, you can use the **GetBiDocs** function to create a blank BiDocs structure. To create the XML structure within that BiDocs, use **CreateElement** to create an XML element, **AddComment** to add an XML comment, **AddAttribute** to add an attribute to an XML element, and **AddProcessInstruction** to add a processing instruction (the first tag of the XML response). Use the inbound Business Interlink function **GenXMLString** to create an XML string from the BiDocs structure. You can use the Write Response object method to send the string as an XML response.

The rest of this section describes the PeopleCode methods and properties you use with the inbound Business Interlink; they parse an XML request contained within a BiDocs object and build an XML response within a BiDocs object. These are all BiDocs methods. There are also functions that you use.

See Chapter 20, "Creating an Inbound Business Interlink," GetBiDoc, page 357 and Chapter 20, "Creating an Inbound Business Interlink," GetContentBody, page 358.

Inbound Business Interlink Methods

This section lists the inbound Business Interlink methods, in alphabetical order.

AddAttribute

Syntax

AddAttribute(*attributename, attributevalue*)

Description

The **AddAttribute** method adds an attribute name and its value to an XML element referenced by a BiDocs object.

Parameters

Parameter	Description
attributename	String. The name of the attribute.
attributevalue	String. The value of the attribute.

Return Value

Number. The return status. NoError, or 0, means the method succeeded.

See Chapter 6, "Business Interlink Class Methods and Properties," GetStatus, page 81.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <candidate>
         <user>
            <location scenery="great" density="low" blank="eh?">
            </location>
```

```
            </user>
         </candidate>
      </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &candidates, &location, &user;
Local number &ret;

&rootDoc = GetBiDoc("");
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&candidates = &postreqresponse.CreateElement("candidates");
&user = &candidates.CreateElement("user");
&location = &user.CreateElement("location");
&ret = &location.AddAttribute("scenery", "great");
&ret = &location.AddAttribute("density", "low");
&ret = &location.AddAttribute("blank", "eh?");
```

AddComment

Syntax

AddComment(*comment*)

Description

The **AddComment** method adds an XML comment after the beginning tag of an XML element referenced by a BiDoc object.

Parameters

Parameter	Description
comment	String. The comment.

Return Value

Number. The return status. NoError, or 0, means the method succeeded.

See Chapter 6, “Business Interlink Class Methods and Properties,” GetStatus, page 81.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <!--this is a comment line-->
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopeCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse, &error, &errorcode, &errortext;
Local string &blob;
Local number &ret;
&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&ret = &error.AddComment("this is a comment line");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

AddProcessInstruction

Syntax

AddProcessInstruction(*instruction*)

Description

The **AddProcessInstruction** method adds an XML processing instruction to a BiDocs object. Use this method at the root level of the BiDocs object for inbound Business Interlinks before you add anything else to the BiDocs object.

Parameters

Parameter	Description
instruction	String. The processing instruction.

Return Value

Number. The return status. NoError, or 0, means the method succeeded.

See Chapter 6, “Business Interlink Class Methods and Properties,” GetStatus, page 81.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <!--this is a comment line-->
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopeCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local number &ret;
&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&ret = &error.AddComment("this is a comment line");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

AddText

Syntax

AddText(*text*)

Description

The **AddText** method adds text to an XML element referenced by a BiDocs object.

Parameters

Parameter	Description
text	String. The text.

Return Value

Number. The return status. NoError, or 0, means the method succeeded.

See Chapter 6, “Business Interlink Class Methods and Properties,” GetStatus, page 81.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <!--this is a comment line-->
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopeCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local number &ret;
```

```
&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&ret = &error.AddComment("this is a comment line");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

CreateElement

Syntax

CreateElement(*elementname*)

Description

The **CreateElement** method creates an XML element with the given name within a BiDoc object.

Parameters

Parameter	Description
elementname	String. The XML element name.

Return Value

BiDocs. The reference to the created element.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopeCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local number &ret;

&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
```

```
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

GenXMLString

Syntax

GenXMLString()

Description

The **GenXMLString** method creates an XML string from a BiDocs object. The BiDocs object must contain the shape and data needed for an XML string. This is part of the inbound Business Interlink functionality, which allow PeopleCode to receive an XML request and return an XML response.

Parameters

None.

Return Value

String. The XML string containing the shape and data of the BiDocs object. For example, you can use this method to create an XML string containing an XML response.

Example

The following example takes a BiDocs structure that contains an XML response and puts that into a text string. Once this is done, the %Response.Write function can send this as an XML response.

```
Local BIDocs &rootDoc;
Local string &xmlString;

/* Create a BiDoc structure containing the data and shape of your XML response?
 (code not shown) */

/* Generate the XML string containing the data and shape of your XML?
 response from the BiDoc structure */
&xmlString = &rootDoc.GenXMLString();
%Response.Write(&xmlString);
```

GetAttributeName

Syntax

GetAttributeName(*attributenumber*)

Description

The **GetAttributeName** method gets the name of an attribute within an XML element referenced by a BiDocs object.

Parameters

Parameter	Description
attributenumber	Number. The index number of the attribute.

Return Value

String. The name of the attribute.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <location scenery="great" density="low" blank="eh?">
         <city>San Rafael</city>
         <state>CA</state>
         <zip>94522</zip>
         <country>US</country>
      </location>
   </postreq>
```

Here is the PeopleCode that gets the name of the second attribute in the location node. &attrName is density.

```
Local BIDocs &rootInDoc, postreqDoc, &locationDoc;
Local string &blob, &attrName;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&locationDoc = &postreqDoc.GetNode("location");
&attrName = &locationDoc.GetAttributeName(2);
```

GetAttributeValue

Syntax

GetAttributeValue({*attributename* | *attributenumber*})

Description

The **GetAttributeValue** method gets the value of an attribute within an XML element referenced by a BiDocs object.

Parameters

Parameter	Description
attributenumber | attributename	Specify the attribute you want to get the value for. You can specify either the attribute number (1 for the first attribute, 2 for the second, and so on) or the attribute name (an XML tag.)

Return Value

String. The value of the attribute.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <location scenery="great" density="low" blank="eh?">
         <city>San Rafael</city>
         <state>CA</state>
         <zip>94522</zip>
         <country>US</country>
      </location>
   </postreq>
```

Here is the PeopleCode that gets the value of the second attribute in the location node. &attrValue is low.

```
Local BIDocs &rootInDoc, &postreqDoc, &locationDoc;
Local string &blob, &attrValue;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&locationDoc = &postreqDoc.GetNode("location");
&attrValue = &locationDoc.GetAttributeValue(2);
```

GetNode

Syntax

`GetNode({`*`nodename`* `|` *`nodenumber`*`})`

Description

The **GetNode** method returns a BiDocs reference to a child XML node (element or comment). Use the **GetNode** method upon a BiDocs reference to an XML element in order to access the child nodes for that element.

Parameters

Parameter	Description
nodenumber | nodename	Specify the node you want to reference. You can specify either a node number (the first node is 1, the second 2, and so on) or a node name (that is, the XML tag.)

Return Value

BiDocs. The returned XML element within a BiDocs object.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets the postreqDoc element and the projtitle element.

```
Local BIDocs &rootInDoc, &postreqDoc, &projtitleDoc;
Local string &name, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&projtitleDoc = &postreqDoc.GetNode("projtitle");
```

Here is the PeopleCode that gets the postreqDoc element, the projtitle element, and the third projsubtitle element.

```
Local BIDocs &rootInDoc, &postreqDoc, &projtitleDoc, &projsubtitleDoc;
Local string &name, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode(1);
&projtitleDoc = &postreqDoc.GetNode(2);
&projsubtitleDoc = &projtitleDoc.GetNode(3);
```

In order to parse a list of elenents like <projsubtitle>, where elements have the same name, you must call **GetNode** using an index number rather than the element name.

ParseXMLString

Syntax

ParseXMLString(*XMLstring*)

Description

The **ParseXMLString** method fills an existing BiDocs object with the data and shape from an XML string. This is part of the inbound Business Interlink functionality, which allow PeopleCode to receive an XML request and return an XML response.

Parameters

Parameter	Description
XMLstring	A string containing an XML document.

Return Values

Number. The return status. NoError, or 0, means the method succeeded.

See Chapter 6, “Business Interlink Class Methods and Properties,” GetStatus, page 81.

Example

The following example gets an XML request, creates an empty BiDoc, and then fills the BiDoc with the data and shape contained in the XML string. Once this is done, the GetDoc method and the GetValue method can get the value of the skills XML element, which is contained within the postreq element in the XML request.

```
Local BIDocs &rootInDoc, &postreqDoc;
Local string &blob;
Local number &ret;

&blob = %Request.GetContentBody();
/* process the inbound xml(request)- Create a BiDoc and fill with the request*/
&rootInDoc = GetBiDoc("");
&ret = &rootInDoc.ParseXMLString(&blob);
&postreqDoc = &rootInDoc.GetDoc("postreq");
&ret = &postreqDoc.GetValue("skills", &skills);
```

Inbound Business Interlink Properties

This section discusses the inbound Business Interlink properties, in alphabetical order.

AttributeCount

Description

The **AttributeCount** property gets the number of attributes within an XML element referenced by a BiDocs object.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <location scenery="great" density="low" blank="eh?">
         <city>San Rafael</city>
         <state>CA</state>
         <zip>94522</zip>
         <country>US</country>
      </location>
   </postreq>
```

Here is the PeopleCode that gets the number of attributes in the location XML element. &count should be 3, for scenery, density, and blank.

```
Local BIDocs &rootInDoc, &postreqDoc, &locationDoc;
Local string &blob;
```

```
Local number &count;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&locationDoc = &postreqDoc.GetNode("location");
&count = &locationDoc.AttributeCount;
```

ChildNodeCount

Description

The **ChildNodeCount** property returns the number of XML child nodes within the element referenced by the BiDocs object. Child nodes include XML elements, comments, and processing instructions.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <!--this is a comment line-->
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the XML code that gets the number of nodes within <postreq> and <projtitle>. &count1 is 2, for <email> and <projtitle>, and &count2 is 4, for the three <projsubtitle> nodes and the comment node.

```
Local BIDocs &rootInDoc, &projtitleDoc;
Local string &blob;
Local number &count1, &count2;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&count1 = &postreqDoc.ChildNodeCount;
&projtitleDoc = &postreqDoc.GetNode("projtitle");
&count2 = &projtitleDoc.ChildNodeCount;
```

NodeName

Description

The **NodeName** property gets the name of an XML element referenced by a BiDocs object. Use this to get the name of an XML element when you used GetNode with an index number to retrieve it (meaning that you did not have the name of the XML element when you used GetNode).

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets the name of the <email> element, email.

```
Local BIDocs &rootInDoc, &postreqDoc, &emailDoc;
Local string &emailName, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode(1);
&emailDoc = &postreqDoc.GetNode(1);
&emailName = &emailDoc.NodeName;
```

NodeType

Description

The **NodeType** property gets the type of an XML element referenced by a BiDocs object: 1 is an XML element, 7 is a processing instruction, and 8 is a comment.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <!--this is a comment-->
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets types: &xmlprocType is 7 for processing instruction, postreqDoc is 1 for element, and commentType is 8 for comment.

```
Local BIDocs &rootInDoc, &postreqDoc, &commentDoc;
```

```
Local number &xmlprocType, &postreqType, &commentDoc;
Local string &blob;
&blob = %Request.GetContentBody();

/* <?xml version="1.0"?> */
&rootInDoc = GetBiDoc(&blob);
&xmlprocType = &rootInDoc.NodeType;

/* <postreq> */
&postreqDoc = &rootInDoc.GetNode(1);
&postreqType = &postreqDoc.NodeType;

/* <!--this is a comment--> */
&commentDoc = &postreqDoc.GetNode(2);
&commentType = &commentDoc.NodeType;
```

NodeValue

Syntax

```
NodeValue
```

Description

The **NodeValue** property gets the value of an XML element referenced by a BiDocs object.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets the value of the third <projsubtitle> element, third_subtitle.

```
Local BIDocs &rootInDoc, &postreqDoc, &projtitleDoc, &projsubtitleDoc;
Local string &name, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode(1);
&projtitleDoc = &postreqDoc.GetNode(2);
&projsubtitleDoc = &projtitleDoc.GetNode(3);
&projsubtitleName = &projsubtitleDoc.NodeValue;
```

Inbound Business Interlink Functions

This section lists the inbound Business Interlink functions.

GetBiDoc

Syntax

GetBiDoc(*XMLstring*)

The **GetBiDoc** function creates a BiDocs object. You can populate the BiDoc with the shape and data from *XMLstring*. This is part of the inbound Business Interlink functionality, which allow PeopleCode to receive an XML request and return an XML response.

Parameters

Parameter	Description
XMLstring	A string containing an XML document. You can specify a NULL string for this parameter, that is, two quotation marks ("") without a space between them.

Return Value

BiDocs. The BiDocs structure containing the shape and data of the XML request.

Example

The following example gets an XML request, puts it into a text string, and puts that into a BiDoc. Once this is done, the GetDoc method and the GetValue method can get the value of the skills XML element, which is contained within the postreq XML element in the XML request.

```
Local BIDocs &rootInDoc, &postreqDoc;
Local string &blob;
Local number &ret;

&blob = %Request.GetContentBody();
/* process the inbound xml(request)- Create a BiDoc and fill with the request*/
&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetDoc("postreq");
&ret = &postreqDoc.GetValue("skills", &skills);
```

You can also create an empty BiDoc with GetBiDoc, as in the following example.

```
Local BIDocs &rootInDoc, &postreqDoc;
Local string &blob;
Local number &ret;

&blob = %Request.GetContentBody();
/* process the inbound xml(request)- Create a BiDoc and fill with the request*/
&rootInDoc = GetBiDoc("");
&ret = &rootInDoc.ParseXMLString(&blob);
&postreqDoc = &rootInDoc.GetDoc("postreq");
&ret = &postreqDoc.GetValue("skills", &skills);
```

iScript Methods

This section describes the iScript methods.

GetContentBody

Syntax

```
GetContentBody()
```

Description

The **GetContentBody** Request method retrieves the text content of an XML request. This is part of the inbound Business Interlink functionality, which allow PeopleCode to receive an XML request and return an XML response.

Parameters

None.

Return Value

String. The received XML request.

Example

The following example gets an XML request, puts it into a text string, and puts that into a BiDoc. Once this is done, the GetDoc method and the GetValue method can get the value of the skills XML element, which is contained within the postreq XML element in the XML request.

```
Local BIDocs &rootInDoc, &postreqDoc;
Local string &blob;
Local number &ret;

&blob = %Request.GetContentBody();
/* process the inbound xml(request)- Create a BiDoc */
&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetDoc("postreq");
&ret = &postreqDoc.GetValue("skills", &skills);
```

APPENDIX A

ISO Country and Currency Codes

PeopleBooks use International Organization for Standardization (ISO) country and currency codes to identify country-specific information and monetary amounts.

This appendix discusses:

- ISO country codes.
- ISO currency codes.

See Also

"About This PeopleBook," Typographical Conventions and Visual Cues

ISO Country Codes

This table lists the ISO country codes that may appear as country identifiers in PeopleBooks:

ISO Country Code	Country Name
ABW	Aruba
AFG	Afghanistan
AGO	Angola
AIA	Anguilla
ALB	Albania
AND	Andorra
ANT	Netherlands Antilles
ARE	United Arab Emirates
ARG	Argentina
ARM	Armenia
ASM	American Samoa
ATA	Antarctica

ISO Country Code	Country Name
ATF	French Southern Territories
ATG	Antigua and Barbuda
AUS	Australia
AUT	Austria
AZE	Azerbaijan
BDI	Burundi
BEL	Belgium
BEN	Benin
BFA	Burkina Faso
BGD	Bangladesh
BGR	Bulgaria
BHR	Bahrain
BHS	Bahamas
BIH	Bosnia and Herzegovina
BLR	Belarus
BLZ	Belize
BMU	Bermuda
BOL	Bolivia
BRA	Brazil
BRB	Barbados
BRN	Brunei Darussalam
BTN	Bhutan
BVT	Bouvet Island
BWA	Botswana
CAF	Central African Republic
CAN	Canada
CCK	Cocos (Keeling) Islands

ISO Country Code	Country Name
CHE	Switzerland
CHL	Chile
CHN	China
CIV	Cote D'Ivoire
CMR	Cameroon
COD	Congo, The Democratic Republic
COG	Congo
COK	Cook Islands
COL	Colombia
COM	Comoros
CPV	Cape Verde
CRI	Costa Rica
CUB	Cuba
CXR	Christmas Island
CYM	Cayman Islands
CYP	Cyprus
CZE	Czech Republic
DEU	Germany
DJI	Djibouti
DMA	Dominica
DNK	Denmark
DOM	Dominican Republic
DZA	Algeria
ECU	Ecuador
EGY	Egypt
ERI	Eritrea
ESH	Western Sahara

ISO Country Code	Country Name
ESP	Spain
EST	Estonia
ETH	Ethiopia
FIN	Finland
FJI	Fiji
FLK	Falkland Islands (Malvinas)
FRA	France
FRO	Faroe Islands
FSM	Micronesia, Federated States
GAB	Gabon
GBR	United Kingdom
GEO	Georgia
GHA	Ghana
GIB	Gibraltar
GIN	Guinea
GLP	Guadeloupe
GMB	Gambia
GNB	Guinea-Bissau
GNQ	Equatorial Guinea
GRC	Greece
GRD	Grenada
GRL	Greenland
GTM	Guatemala
GUF	French Guiana
GUM	Guam
GUY	Guyana
GXA	GXA - GP Core Country

ISO Country Code	Country Name
GXB	GXB - GP Core Country
GXC	GXC - GP Core Country
GXD	GXD - GP Core Country
HKG	Hong Kong
HMD	Heard and McDonald Islands
HND	Honduras
HRV	Croatia
HTI	Haiti
HUN	Hungary
IDN	Indonesia
IND	India
IOT	British Indian Ocean Territory
IRL	Ireland
IRN	Iran (Islamic Republic Of)
IRQ	Iraq
ISL	Iceland
ISR	Israel
ITA	Italy
JAM	Jamaica
JOR	Jordan
JPN	Japan
KAZ	Kazakstan
KEN	Kenya
KGZ	Kyrgyzstan
KHM	Cambodia
KIR	Kiribati
KNA	Saint Kitts and Nevis

ISO Country Code	Country Name
KOR	Korea, Republic of
KWT	Kuwait
LAO	Lao People’s Democratic Rep
LBN	Lebanon
LBR	Liberia
LBY	Libyan Arab Jamahiriya
LCA	Saint Lucia
LIE	Liechtenstein
LKA	Sri Lanka
LSO	Lesotho
LTU	Lithuania
LUX	Luxembourg
LVA	Latvia
MAC	Macao
MAR	Morocco
MCO	Monaco
MDA	Moldova, Republic of
MDG	Madagascar
MDV	Maldives
MEX	Mexico
MHL	Marshall Islands
MKD	Fmr Yugoslav Rep of Macedonia
MLI	Mali
MLT	Malta
MMR	Myanmar
MNG	Mongolia
MNP	Northern Mariana Islands

ISO Country Code	Country Name
MOZ	Mozambique
MRT	Mauritania
MSR	Montserrat
MTQ	Martinique
MUS	Mauritius
MWI	Malawi
MYS	Malaysia
MYT	Mayotte
NAM	Namibia
NCL	New Caledonia
NER	Niger
NFK	Norfolk Island
NGA	Nigeria
NIC	Nicaragua
NIU	Niue
NLD	Netherlands
NOR	Norway
NPL	Nepal
NRU	Nauru
NZL	New Zealand
OMN	Oman
PAK	Pakistan
PAN	Panama
PCN	Pitcairn
PER	Peru
PHL	Philippines
PLW	Palau

ISO Country Code	Country Name
PNG	Papua New Guinea
POL	Poland
PRI	Puerto Rico
PRK	Korea, Democratic People's Rep
PRT	Portugal
PRY	Paraguay
PSE	Palestinian Territory, Occupie
PYF	French Polynesia
QAT	Qatar
REU	Reunion
ROU	Romania
RUS	Russian Federation
RWA	Rwanda
SAU	Saudi Arabia
SDN	Sudan
SEN	Senegal
SGP	Singapore
SGS	Sth Georgia & Sth Sandwich Is
SHN	Saint Helena
SJM	Svalbard and Jan Mayen
SLB	Solomon Islands
SLE	Sierra Leone
SLV	El Salvador
SMR	San Marino
SOM	Somalia
SPM	Saint Pierre and Miquelon
STP	Sao Tome and Principe

ISO Country Code	Country Name
SUR	Suriname
SVK	Slovakia
SVN	Slovenia
SWE	Sweden
SWZ	Swaziland
SYC	Seychelles
SYR	Syrian Arab Republic
TCA	Turks and Caicos Islands
TCD	Chad
TGO	Togo
THA	Thailand
TJK	Tajikistan
TKL	Tokelau
TKM	Turkmenistan
TLS	East Timor
TON	Tonga
TTO	Trinidad and Tobago
TUN	Tunisia
TUR	Turkey
TUV	Tuvalu
TWN	Taiwan, Province of China
TZA	Tanzania, United Republic of
UGA	Uganda
UKR	Ukraine
UMI	US Minor Outlying Islands
URY	Uruguay
USA	United States

ISO Country Code	Country Name
UZB	Uzbekistan
VAT	Holy See (Vatican City State)
VCT	St Vincent and the Grenadines
VEN	Venezuela
VGB	Virgin Islands (British)
VIR	Virgin Islands (U.S.)
VNM	Viet Nam
VUT	Vanuatu
WLF	Wallis and Futuna Islands
WSM	Samoa
YEM	Yemen
YUG	Yugoslavia
ZAF	South Africa
ZMB	Zambia
ZWE	Zimbabwe

ISO Currency Codes

This table lists the ISO country codes that may appear as currency identifiers in PeopleBooks:

ISO Currency Code	Description
ADP	Andorran Peseta
AED	United Arab Emirates Dirham
AFA	Afghani
AFN	Afghani
ALK	Old Lek
ALL	Lek
AMD	Armenian Dram

ISO Currency Code	Description
ANG	Netherlands Antilles Guilder
AOA	Kwanza
AOK	Kwanza
AON	New Kwanza
AOR	Kwanza Reajustado
ARA	Austral
ARP	Peso Argentino
ARS	Argentine Peso
ARY	Peso
ATS	Schilling
AUD	Australian Dollar
AWG	Aruban Guilder
AZM	Azerbaijanian Manat
BAD	Dinar
BAM	Convertible Marks
BBD	Barbados Dollar
BDT	Taka
BEC	Convertible Franc
BEF	Belgian Franc
BEL	Financial Belgian Franc
BGJ	Lev A/52
BGK	Lev A/62
BGL	Lev
BGN	Bulgarian LEV
BHD	Bahraini Dinar
BIF	Burundi Franc
BMD	Bermudian Dollar

ISO Currency Code	Description
BND	Brunei Dollar
BOB	Boliviano
BOP	Peso
BOV	Mvdol
BRB	Cruzeiro
BRC	Cruzado
BRE	Cruzeiro
BRL	Brazilian Real
BRN	New Cruzado
BRR	Brazilian Real Dollar
BSD	Bahamian Dollar
BTN	Ngultrum
BUK	N/A
BWP	Pula
BYB	Belarussian Ruble
BYR	Belarussian Ruble
BZD	Belize Dollar
CAD	Canadian Dollar
CDF	Franc Congolais
CHF	Swiss Franc
CLF	Unidades de fomento
CLP	Chilean Peso
CNX	Peoples Bank Dollar
CNY	Yuan Renminbi
COP	Colombian Peso
CRC	Costa Rican Colon
CSD	Serbia Dinar

ISO Currency Code	Description
CSJ	Krona A/53
CSK	Koruna
CUP	Cuban Peso
CVE	Cape Verde Escudo
CYP	Cyprus Pound
CZK	Czech Koruna
DEM	Deutsche Mark
DJF	Djibouti Franc
DKK	Danish Krone
DOP	Dominican Peso
DZD	Algerian Dinar
ECS	Sucre
ECV	Unidad de Valor
EEK	Kroon
EGP	Egyptian Pound
EQE	Ekwele
ERN	Nakfa
ESA	Spanish Peseta
ESB	Convertible Peseta
ESP	Spanish Peseta
ETB	Ethiopian Birr
EUR	euro
FIM	Markka
FJD	Fiji Dollar
FKP	Falklands Isl. Pound
FRF	French Franc
GBP	Pound Sterling

ISO Currency Code	Description
GEK	Georgian Coupon
GEL	Lari
GHC	Cedi
GIP	Gibraltar Pound
GMD	Dalasi
GNE	Syli
GNF	Guinea Franc
GNS	Syli
GQE	Ekwele
GRD	Drachma
GTQ	Quetzal
GWE	Guinea Escudo
GWP	Guinea-Bissau Peso
GYD	Guyana Dollar
HKD	Hong Kong Dollar
HNL	Lempira
HRD	Dinar
HRK	Kuna
HTG	Gourde
HUF	Forint
IDR	Rupiah
IEP	Irish Pound
ILP	Pound
ILR	Old Shekel
ILS	New Israeli Sheqel
INR	Indian Rupee
IQD	Iraqi Dinar

ISO Currency Code	Description
IRR	Iranian Rial
ISJ	Old Krona
ISK	Iceland Krona
ITL	Italian Lira
JMD	Jamaican Dollar
JOD	Jordanian Dinar
JPY	Yen
KES	Kenyan Shilling
KGS	Som
KHR	Riel
KMF	Comoro Franc
KPW	North Korean Won
KRW	Won
KWD	Kuwaiti Dinar
KYD	Cayman Islands dollar
KZT	Tenge
LAJ	Kip Pot Pol
LAK	Kip
LBP	Lebanese Pound
LKR	Sri Lanka Rupee
LRD	Liberian Dollar
LSL	Loti
LSM	Maloti
LTL	Lithuanian Litas
LTT	Talonas
LUC	Convertib Francl
LUF	Luxembourg Franc

ISO Currency Code	Description
LUL	Financial Franc
LVL	Latvian Lats
LVR	Latvian Ruble
LYD	Libyan Dinar
MAD	Moroccan Dirham
MAF	Mali Franc
MDL	Moldovan Leu
MGF	Malagasy Franc
MKD	Denar
MLF	Mali Franc
MMK	Kyat
MNT	Tugrik
MOP	Pataca
MRO	Ouguiya
MTL	Maltese Lira
MTP	Maltese Pound
MUR	Mauritius Rupee
MVQ	Maldive Rupee
MVR	Rufiyaa
MWK	Malawian Kwacha
MXN	Mexican Peso
MXP	Mexican Peso
MXV	Mexican UDI
MYR	Malaysian Ringgit
MZE	Mozambique Escudo
MZM	Metical
NAD	Namibia Dollar

ISO Currency Code	Description
NGN	Naira
NIC	Cordoba
NIO	Cordoba Oro
NLG	Netherlands Guilder
NOK	Norwegian Krone
NPR	Nepalese Rupee
NZD	New Zealand Dollar
OMR	Rial Omani
PAB	Balboa
PEI	Inti
PEN	Nuevo Sol
PES	Sol
PGK	Kina
PHP	Philippine Peso
PKR	Pakistan Rupee
PLN	Zloty
PLZ	Zloty
PTE	Portuguese Escudo
PYG	Guarani
QAR	Qatari Rial
ROK	Leu A/52
ROL	Leu
RUB	Russian Ruble
RUR	Russian Federation Rouble
RWF	Rwanda Franc
SAR	Saudi Riyal
SBD	Solomon Islands

ISO Currency Code	Description
SCR	Seychelles Rupee
SDD	Sudanese Dinar
SDP	Sudanese Pound
SEK	Swedish Krona
SGD	Singapore Dollar
SHP	St Helena Pound
SIT	Tolar
SKK	Slovak Koruna
SLL	Leone
SOS	Somali Shilling
SRG	Surinam Guilder
STD	Dobra
SUR	Rouble
SVC	El Salvador Colon
SYP	Syrian Pound
SZL	Lilangeni
THB	Baht
TJR	Tajik Ruble
TJS	Somoni
TMM	Manat
TND	Tunisian Dinar
TOP	Pa’anga
TPE	Timor Escudo
TRL	Turkish Lira
TTD	Trinidad Dollar
TWD	New Taiwan Dollar
TZS	Tanzanian Shilling

ISO Currency Code	Description
UAH	Hryvnia
UAK	Karbovanet
UGS	Uganda Shilling
UGW	Old Shilling
UGX	Uganda Shilling
USD	US Dollar
USN	US Dollar (Next day)
USS	US Dollar (Same day)
UYN	Old Uruguay Peso
UYP	Uruguayan Peso
UYU	Peso Uruguayo
UZS	Uzbekistan Sum
VEB	Bolivar
VNC	Old Dong
VND	Dong
VUV	Vatu
WST	Tala
XAF	CFA Franc BEAC
XAG	Silver
XAU	GOLD
XBA	European Composite Unit
XBB	European Monetary Unit
XBC	European Unit of Account 9
XBD	European Unit of Account 17
XCD	East Caribbean Dollar
XDR	SDR
XEU	EU Currency (E.C.U)

ISO Currency Code	Description
XFO	Gold-Franc
XFU	UIC-Franc
XOF	CFA Franc BCEAO
XPD	Palladium
XPF	CFP Franc
XPT	Platinum
XTS	For Testing Purposes
XXX	Non Currency Transaction
YDD	Yemeni Din
YER	Yemeni Rial
YUD	New Yugoslavian Dinar
YUM	New Dinar
YUN	Yugoslavian Dinar
ZAL	Financial Rand
ZAR	Rand
ZMK	Zambian Kwacha
ZRN	New Zaire
ZRZ	Zaire
ZWC	Rhodesian Dollar
ZWD	Zimbabwe Dollar

Glossary of PeopleSoft Terms

absence entitlement This element defines rules for granting paid time off for valid absences, such as sick time, vacation, and maternity leave. An absence entitlement element defines the entitlement amount, frequency, and entitlement period.

absence take This element defines the conditions that must be met before a payee is entitled to take paid time off.

accounting class In PeopleSoft Enterprise Performance Management, the accounting class defines how a resource is treated for generally accepted accounting practices. The Inventory class indicates whether a resource becomes part of a balance sheet account, such as inventory or fixed assets, while the Non-inventory class indicates that the resource is treated as an expense of the period during which it occurs.

accounting date The accounting date indicates when a transaction is recognized, as opposed to the date the transaction actually occurred. The accounting date and transaction date can be the same. The accounting date determines the period in the general ledger to which the transaction is to be posted. You can only select an accounting date that falls within an open period in the ledger to which you are posting. The accounting date for an item is normally the invoice date.

accounting split The accounting split method indicates how expenses are allocated or divided among one or more sets of accounting ChartFields.

accumulator You use an accumulator to store cumulative values of defined items as they are processed. You can accumulate a single value over time or multiple values over time. For example, an accumulator could consist of all voluntary deductions, or all company deductions, enabling you to accumulate amounts. It allows total flexibility for time periods and values accumulated.

action reason The reason an employee's job or employment information is updated. The action reason is entered in two parts: a personnel action, such as a promotion, termination, or change from one pay group to another—and a reason for that action. Action reasons are used by PeopleSoft Human Resources, PeopleSoft Benefits Administration, PeopleSoft Stock Administration, and the COBRA Administration feature of the Base Benefits business process.

action template In PeopleSoft Receivables, outlines a set of escalating actions that the system or user performs based on the period of time that a customer or item has been in an action plan for a specific condition.

activity In PeopleSoft Enterprise Learning Management, an instance of a catalog item (sometimes called a class) that is available for enrollment. The activity defines such things as the costs that are associated with the offering, enrollment limits and deadlines, and waitlisting capacities.

In PeopleSoft Enterprise Performance Management, the work of an organization and the aggregation of actions that are used for activity-based costing.

In PeopleSoft Project Costing, the unit of work that provides a further breakdown of projects—usually into specific tasks.

In PeopleSoft Workflow, a specific transaction that you might need to perform in a business process. Because it consists of the steps that are used to perform a transaction, it is also known as a step map.

agreement In PeopleSoft eSettlements, provides a way to group and specify processing options, such as payment terms, pay from a bank, and notifications by a buyer and supplier location combination.

allocation rule In PeopleSoft Enterprise Incentive Management, an expression within compensation plans that enables the system to assign transactions to nodes and participants. During transaction allocation, the allocation engine traverses the compensation structure from the current node to the root node, checking each node for plans that contain allocation rules.

alternate account A feature in PeopleSoft General Ledger that enables you to create a statutory chart of accounts and enter statutory account transactions at the detail transaction level, as required for recording and reporting by some national governments.

AR specialist Abbreviation for *receivables specialist.* In PeopleSoft Receivables, an individual in who tracks and resolves deductions and disputed items.

arbitration plan In PeopleSoft Enterprise Pricer, defines how price rules are to be applied to the base price when the transaction is priced.

assessment rule In PeopleSoft Receivables, a user-defined rule that the system uses to evaluate the condition of a customer's account or of individual items to determine whether to generate a follow-up action.

asset class An asset group used for reporting purposes. It can be used in conjunction with the asset category to refine asset classification.

attribute/value pair In PeopleSoft Directory Interface, relates the data that makes up an entry in the directory information tree.

authentication server A server that is set up to verify users of the system.

base time period In PeopleSoft Business Planning, the lowest level time period in a calendar.

benchmark job In PeopleSoft Workforce Analytics, a benchmark job is a job code for which there is corresponding salary survey data from published, third-party sources.

book In PeopleSoft Asset Management, used for storing financial and tax information, such as costs, depreciation attributes, and retirement information on assets.

branch A tree node that rolls up to nodes above it in the hierarchy, as defined in PeopleSoft Tree Manager.

budgetary account only An account used by the system only and not by users; this type of account does not accept transactions. You can only budget with this account. Formerly called "system-maintained account."

budget check In commitment control, the processing of source transactions against control budget ledgers, to see if they pass, fail, or pass with a warning.

budget control In commitment control, budget control ensures that commitments and expenditures don't exceed budgets. It enables you to track transactions against corresponding budgets and terminate a document's cycle if the defined budget conditions are not met. For example, you can prevent a purchase order from being dispatched to a vendor if there are insufficient funds in the related budget to support it.

budget period The interval of time (such as 12 months or 4 quarters) into which a period is divided for budgetary and reporting purposes. The ChartField allows maximum flexibility to define operational accounting time periods without restriction to only one calendar.

business event In PeopleSoft Receivables, defines the processing characteristics for the Receivable Update process for a draft activity.

	In PeopleSoft Sales Incentive Management, an original business transaction or activity that may justify the creation of a PeopleSoft Enterprise Incentive Management event (a sale, for example).
business unit	A corporation or a subset of a corporation that is independent with regard to one or more operational or accounting functions.
buyer	In PeopleSoft eSettlements, an organization (or business unit, as opposed to an individual) that transacts with suppliers (vendors) within the system. A buyer creates payments for purchases that are made in the system.
catalog item	In PeopleSoft Enterprise Learning Management, a specific topic that a learner can study and have tracked. For example, “Introduction to Microsoft Word.” A catalog item contains general information about the topic and includes a course code, description, categorization, keywords, and delivery methods. A catalog item can have one or more learning activities.
catalog map	In PeopleSoft Catalog Management, translates values from the catalog source data to the format of the company’s catalog.
catalog partner	In PeopleSoft Catalog Management, shares responsibility with the enterprise catalog manager for maintaining catalog content.
categorization	Associates partner offerings with catalog offerings and groups them into enterprise catalog categories.
channel	In PeopleSoft MultiChannel Framework, email, chat, voice (computer telephone integration [CTI]), or a generic event.
ChartField	A field that stores a chart of accounts, resources, and so on, depending on the PeopleSoft application. ChartField values represent individual account numbers, department codes, and so forth.
ChartField balancing	You can require specific ChartFields to match up (balance) on the debit and the credit side of a transaction.
ChartField combination edit	The process of editing journal lines for valid ChartField combinations based on user-defined rules.
ChartKey	One or more fields that uniquely identify each row in a table. Some tables contain only one field as the key, while others require a combination.
checkbook	In PeopleSoft Promotions Management, enables you to view financial data (such as planned, incurred, and actual amounts) that is related to funds and trade promotions.
Class ChartField	A ChartField value that identifies a unique appropriation budget key when you combine it with a fund, department ID, and program code, as well as a budget period. Formerly called *sub-classification.*
clone	In PeopleCode, to make a unique copy. In contrast, to *copy* may mean making a new reference to an object, so if the underlying object is changed, both the copy and the original change.
collection	To make a set of documents available for searching in Verity, you must first create at least one collection. A collection is set of directories and files that allow search application users to use the Verity search engine to quickly find and display source documents that match search criteria. A collection is a set of statistics and pointers to the source documents, stored in a proprietary format on a file server. Because a collection can only store information for a single location, PeopleSoft maintains a set of collections (one per language code) for each search index object.

collection rule In PeopleSoft Receivables, a user-defined rule that defines actions to take for a customer based on both the amount and the number of days past due for outstanding balances.

compensation object In PeopleSoft Enterprise Incentive Management, a node within a compensation structure. Compensation objects are the building blocks that make up a compensation structure's hierarchical representation.

compensation structure In PeopleSoft Enterprise Incentive Management, a hierarchical relationship of compensation objects that represents the compensation-related relationship between the objects.

condition In PeopleSoft Receivables, occurs when there is a change of status for a customer's account, such as reaching a credit limit or exceeding a user-defined balance due.

configuration parameter catalog Used to configure an external system with PeopleSoft. For example, a configuration parameter catalog might set up configuration and communication parameters for an external server.

configuration plan In PeopleSoft Enterprise Incentive Management, configuration plans hold allocation information for common variables (not incentive rules) and are attached to a node without a participant. Configuration plans are not processed by transactions.

content reference Content references are pointers to content registered in the portal registry. These are typically either URLs or iScripts. Content references fall into three categories: target content, templates, and template pagelets.

context In PeopleCode, determines which buffer fields can be contextually referenced and which is the current row of data on each scroll level when a PeopleCode program is running.

In PeopleSoft Enterprise Incentive Management, a mechanism that is used to determine the scope of a processing run. PeopleSoft Enterprise Incentive Management uses three types of context: plan, period, and run-level.

control table Stores information that controls the processing of an application. This type of processing might be consistent throughout an organization, or it might be used only by portions of the organization for more limited sharing of data.

cost profile A combination of a receipt cost method, a cost flow, and a deplete cost method. A profile is associated with a cost book and determines how items in that book are valued, as well as how the material movement of the item is valued for the book.

cost row A cost transaction and amount for a set of ChartFields.

current learning In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's in-progress learning activities and programs.

data acquisition In PeopleSoft Enterprise Incentive Management, the process during which raw business transactions are acquired from external source systems and fed into the operational data store (ODS).

data elements Data elements, at their simplest level, define a subset of data and the rules by which to group them.

For Workforce Analytics, data elements are rules that tell the system what measures to retrieve about your workforce groups.

dataset A data grouping that enables role-based filtering and distribution of data. You can limit the range and quantity of data that is displayed for a user by associating dataset rules with user roles. The result of dataset rules is a set of data that is appropriate for the user's roles.

delivery method In PeopleSoft Enterprise Learning Management, identifies the primary type of delivery method in which a particular learning activity is offered. Also provides default values for the learning activity, such as cost and language. This is primarily used to help learners search the catalog for the type of delivery from which they learn best. Because PeopleSoft Enterprise Learning Management is a blended learning system, it does not enforce the delivery method.

In PeopleSoft Supply Chain Management, identifies the method by which goods are shipped to their destinations (such as truck, air, rail, and so on). The delivery method is specified when creating shipment schedules.

delivery method type In PeopleSoft Enterprise Learning Management, identifies how learning activities can be delivered—for example, through online learning, classroom instruction, seminars, books, and so forth—in an organization. The type determines whether the delivery method includes scheduled components.

directory information tree In PeopleSoft Directory Interface, the representation of a directory's hierarchical structure.

document sequencing A flexible method that sequentially numbers the financial transactions (for example, bills, purchase orders, invoices, and payments) in the system for statutory reporting and for tracking commercial transaction activity.

dynamic detail tree A tree that takes its detail values—dynamic details—directly from a table in the database, rather than from a range of values that are entered by the user.

edit table A table in the database that has its own record definition, such as the Department table. As fields are entered into a PeopleSoft application, they can be validated against an edit table to ensure data integrity throughout the system.

effective date A method of dating information in PeopleSoft applications. You can predate information to add historical data to your system, or postdate information in order to enter it before it actually goes into effect. By using effective dates, you don't delete values; you enter a new value with a current effective date.

EIM ledger Abbreviation for *Enterprise Incentive Management ledger.* In PeopleSoft Enterprise Incentive Management, an object to handle incremental result gathering within the scope of a participant. The ledger captures a result set with all of the appropriate traces to the data origin and to the processing steps of which it is a result.

elimination set In PeopleSoft General Ledger, a related group of intercompany accounts that is processed during consolidations.

entry event In PeopleSoft General Ledger, Receivables, Payables, Purchasing, and Billing, a business process that generates multiple debits and credits resulting from single transactions to produce standard, supplemental accounting entries.

equitization In PeopleSoft General Ledger, a business process that enables parent companies to calculate the net income of subsidiaries on a monthly basis and adjust that amount to increase the investment amount and equity income amount before performing consolidations.

event A predefined point either in the Component Processor flow or in the program flow. As each point is encountered, the event activates each component, triggering any PeopleCode program that is associated with that component and that event. Examples of events are FieldChange, SavePreChange, and RowDelete.

In PeopleSoft Human Resources, also refers to an incident that affects benefits eligibility.

event propagation process In PeopleSoft Sales Incentive Management, a process that determines, through logic, the propagation of an original PeopleSoft Enterprise Incentive Management event and creates a derivative (duplicate) of the original event to be processed by other objects.

	Sales Incentive Management uses this mechanism to implement splits, roll-ups, and so on. Event propagation determines who receives the credit.
exception	In PeopleSoft Receivables, an item that either is a deduction or is in dispute.
exclusive pricing	In PeopleSoft Order Management, a type of arbitration plan that is associated with a price rule. Exclusive pricing is used to price sales order transactions.
fact	In PeopleSoft applications, facts are numeric data values from fields from a source database as well as an analytic application. A fact can be anything you want to measure your business by, for example, revenue, actual, budget data, or sales numbers. A fact is stored on a fact table.
forecast item	A logical entity with a unique set of descriptive demand and forecast data that is used as the basis to forecast demand. You create forecast items for a wide range of uses, but they ultimately represent things that you buy, sell, or use in your organization and for which you require a predictable usage.
fund	In PeopleSoft Promotions Management, a budget that can be used to fund promotional activity. There are four funding methods: top down, fixed accrual, rolling accrual, and zero-based accrual.
generic process type	In PeopleSoft Process Scheduler, process types are identified by a generic process type. For example, the generic process type SQR includes all SQR process types, such as SQR process and SQR report.
group	In PeopleSoft Billing and Receivables, a posting entity that comprises one or more transactions (items, deposits, payments, transfers, matches, or write-offs). In PeopleSoft Human Resources Management and Supply Chain Management, any set of records that are associated under a single name or variable to run calculations in PeopleSoft business processes. In PeopleSoft Time and Labor, for example, employees are placed in groups for time reporting purposes.
incentive object	In PeopleSoft Enterprise Incentive Management, the incentive-related objects that define and support the PeopleSoft Enterprise Incentive Management calculation process and results, such as plan templates, plans, results data, user interaction objects, and so on.
incentive rule	In PeopleSoft Sales Incentive Management, the commands that act on transactions and turn them into compensation. A rule is one part in the process of turning a transaction into compensation.
incur	In PeopleSoft Promotions Management, to become liable for a promotional payment. In other words, you owe that amount to a customer for promotional activities.
item	In PeopleSoft Inventory, a tangible commodity that is stored in a business unit (shipped from a warehouse). In PeopleSoft Demand Planning, Inventory Policy Planning, and Supply Planning, a noninventory item that is designated as being used for planning purposes only. It can represent a family or group of inventory items. It can have a planning bill of material (BOM) or planning routing, and it can exist as a component on a planning BOM. A planning item cannot be specified on a production or engineering BOM or routing, and it cannot be used as a component in a production. The quantity on hand will never be maintained. In PeopleSoft Receivables, an individual receivable. An item can be an invoice, a credit memo, a debit memo, a write-off, or an adjustment.
KPI	An abbreviation for *key performance indicator.* A high-level measurement of how well an organization is doing in achieving critical success factors. This defines the data value or calculation upon which an assessment is determined.

LDIF file Abbreviation for *Lightweight Directory Access Protocol (LDAP) Data Interchange Format file.* Contains discrepancies between PeopleSoft data and directory data.

learner group In PeopleSoft Enterprise Learning Management, a group of learners who are linked to the same learning environment. Members of the learner group can share the same attributes, such as the same department or job code. Learner groups are used to control access to and enrollment in learning activities and programs. They are also used to perform group enrollments and mass enrollments in the back office.

learning components In PeopleSoft Enterprise Learning Management, the foundational building blocks of learning activities. PeopleSoft Enterprise Learning Management supports six basic types of learning components: web-based, session, webcast, test, survey, and assignment. One or more of these learning component types compose a single learning activity.

learning environment In PeopleSoft Enterprise Learning Management, identifies a set of categories and catalog items that can be made available to learner groups. Also defines the default values that are assigned to the learning activities and programs that are created within a particular learning environment. Learning environments provide a way to partition the catalog so that learners see only those items that are relevant to them.

learning history In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's completed learning activities and programs.

ledger mapping You use ledger mapping to relate expense data from general ledger accounts to resource objects. Multiple ledger line items can be mapped to one or more resource IDs. You can also use ledger mapping to map dollar amounts (referred to as *rates*) to business units. You can map the amounts in two different ways: an actual amount that represents actual costs of the accounting period, or a budgeted amount that can be used to calculate the capacity rates as well as budgeted model results. In PeopleSoft Enterprise Warehouse, you can map general ledger accounts to the EW Ledger table.

library section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan (or template) and that is available for other plans to share. Changes to a library section are reflected in all plans that use it.

linked section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan template but appears in a plan. Changes to linked sections propagate to plans using that section.

linked variable In PeopleSoft Enterprise Incentive Management, a variable that is defined and maintained in a plan template and that also appears in a plan. Changes to linked variables propagate to plans using that variable.

load In PeopleSoft Inventory, identifies a group of goods that are shipped together. Load management is a feature of PeopleSoft Inventory that is used to track the weight, the volume, and the destination of a shipment.

local functionality In PeopleSoft HRMS, the set of information that is available for a specific country. You can access this information when you click the appropriate country flag in the global window, or when you access it by a local country menu.

location Locations enable you to indicate the different types of addresses—for a company, for example, one address to receive bills, another for shipping, a third for postal deliveries, and a separate street address. Each address has a different location number. The primary location—indicated by a *1*—is the address you use most often and may be different from the main address.

logistical task In PeopleSoft Services Procurement, an administrative task that is related to hiring a service provider. Logistical tasks are linked to the service type on the work order so that different types of services can have different logistical tasks. Logistical tasks include both preapproval tasks (such as assigning a new badge or ordering a new

	laptop) and postapproval tasks (such as scheduling orientation or setting up the service provider email). The logistical tasks can be mandatory or optional. Mandatory preapproval tasks must be completed before the work order is approved. Mandatory postapproval tasks, on the other hand, must be completed before a work order is released to a service provider.
market template	In PeopleSoft Enterprise Incentive Management, additional functionality that is specific to a given market or industry and is built on top of a product category.
match group	In PeopleSoft Receivables, a group of receivables items and matching offset items. The system creates match groups by using user-defined matching criteria for selected field values.
MCF server	Abbreviation for *PeopleSoft MultiChannel Framework server.* Comprises the universal queue server and the MCF log server. Both processes are started when *MCF Servers* is selected in an application server domain configuration.
merchandising activity	In PeopleSoft Promotions Management, a specific discount type that is associated with a trade promotion (such as off-invoice, billback or rebate, or lump-sum payment) that defines the performance that is required to receive the discount. In the industry, you may know this as an offer, a discount, a merchandising event, an event, or a tactic.
meta-SQL	Meta-SQL constructs expand into platform-specific Structured Query Language (SQL) substrings. They are used in functions that pass SQL strings, such as in SQL objects, the SQLExec function, and PeopleSoft Application Engine programs.
metastring	Metastrings are special expressions included in SQL string literals. The metastrings, prefixed with a percent (%) symbol, are included directly in the string literals. They expand at run time into an appropriate substring for the current database platform.
multibook	In PeopleSoft General Ledger, multiple ledgers having multiple-base currencies that are defined for a business unit, with the option to post a single transaction to all base currencies (all ledgers) or to only one of those base currencies (ledgers).
multicurrency	The ability to process transactions in a currency other than the business unit's base currency.
national allowance	In PeopleSoft Promotions Management, a promotion at the corporate level that is funded by nondiscretionary dollars. In the industry, you may know this as a national promotion, a corporate promotion, or a corporate discount.
node-oriented tree	A tree that is based on a detail structure, but the detail values are not used.
pagelet	Each block of content on the home page is called a pagelet. These pagelets display summary information within a small rectangular area on the page. The pagelet provide users with a snapshot of their most relevant PeopleSoft and non-PeopleSoft content.
participant	In PeopleSoft Enterprise Incentive Management, participants are recipients of the incentive compensation calculation process.
participant object	Each participant object may be related to one or more compensation objects. See also *compensation object.*
partner	A company that supplies products or services that are resold or purchased by the enterprise.
pay cycle	In PeopleSoft Payables, a set of rules that define the criteria by which it should select scheduled payments for payment creation.
pending item	In PeopleSoft Receivables, an individual receivable (such as an invoice, a credit memo, or a write-off) that has been entered in or created by the system, but hasn't been posted.

PeopleCode PeopleCode is a proprietary language, executed by the PeopleSoft application processor. PeopleCode generates results based upon existing data or user actions. By using business interlink objects, external services are available to all PeopleSoft applications wherever PeopleCode can be executed.

PeopleCode event An action that a user takes upon an object, usually a record field, that is referenced within a PeopleSoft page.

PeopleSoft Internet Architecture The fundamental architecture on which PeopleSoft 8 applications are constructed, consisting of a relational database management system (RDBMS), an application server, a web server, and a browser.

performance measurement In PeopleSoft Enterprise Incentive Management, a variable used to store data (similar to an aggregator, but without a predefined formula) within the scope of an incentive plan. Performance measures are associated with a plan calendar, territory, and participant. Performance measurements are used for quota calculation and reporting.

period context In PeopleSoft Enterprise Incentive Management, because a participant typically uses the same compensation plan for multiple periods, the period context associates a plan context with a specific calendar period and fiscal year. The period context references the associated plan context, thus forming a chain. Each plan context has a corresponding set of period contexts.

plan In PeopleSoft Sales Incentive Management, a collection of allocation rules, variables, steps, sections, and incentive rules that instruct the PeopleSoft Enterprise Incentive Management engine in how to process transactions.

plan context In PeopleSoft Enterprise Incentive Management, correlates a participant with the compensation plan and node to which the participant is assigned, enabling the PeopleSoft Enterprise Incentive Management system to find anything that is associated with the node and that is required to perform compensation processing. Each participant, node, and plan combination represents a unique plan context—if three participants are on a compensation structure, each has a different plan context. Configuration plans are identified by plan contexts and are associated with the participants that refer to them.

plan template In PeopleSoft Enterprise Incentive Management, the base from which a plan is created. A plan template contains common sections and variables that are inherited by all plans that are created from the template. A template may contain steps and sections that are not visible in the plan definition.

planned learning In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's planned learning activities and programs.

planning instance In PeopleSoft Supply Planning, a set of data (business units, items, supplies, and demands) constituting the inputs and outputs of a supply plan.

portal registry In PeopleSoft applications, the portal registry is a tree-like structure in which content references are organized, classified, and registered. It is a central repository that defines both the structure and content of a portal through a hierarchical, tree-like structure of folders useful for organizing and securing content references.

price list In PeopleSoft Enterprise Pricer, enables you to select products and conditions for which the price list applies to a transaction. During a transaction, the system either determines the product price based on the predefined search hierarchy for the transaction or uses the product's lowest price on any associated, active price lists. This price is used as the basis for any further discounts and surcharges.

price rule In PeopleSoft Enterprise Pricer, defines the conditions that must be met for adjustments to be applied to the base price. Multiple rules can apply when conditions of each rule are met.

price rule condition In PeopleSoft Enterprise Pricer, selects the price-by fields, the values for the price-by fields, and the operator that determines how the price-by fields are related to the transaction.

price rule key In PeopleSoft Enterprise Pricer, defines the fields that are available to define price rule conditions (which are used to match a transaction) on the price rule.

process category In PeopleSoft Process Scheduler, processes that are grouped for server load balancing and prioritization.

process group In PeopleSoft Financials, a group of application processes (performed in a defined order) that users can initiate in real time, directly from a transaction entry page.

process definition Process definitions define each run request.

process instance A unique number that identifies each process request. This value is automatically incremented and assigned to each requested process when the process is submitted to run.

process job You can link process definitions into a job request and process each request serially or in parallel. You can also initiate subsequent processes based on the return code from each prior request.

process request A single run request, such as a Structured Query Report (SQR), a COBOL or Application Engine program, or a Crystal report that you run through PeopleSoft Process Scheduler.

process run control A PeopleTools variable used to retain PeopleSoft Process Scheduler values needed at runtime for all requests that reference a run control ID. Do not confuse these with application run controls, which may be defined with the same run control ID, but only contain information specific to a given application process request.

product category In PeopleSoft Enterprise Incentive Management, indicates an application in the Enterprise Incentive Management suite of products. Each transaction in the PeopleSoft Enterprise Incentive Management system is associated with a product category.

programs In PeopleSoft Enterprise Learning Management, a high-level grouping that guides the learner along a specific learning path through sections of catalog items. PeopleSoft Enterprise Learning Systems provides two types of programs—curricula and certifications.

progress log In PeopleSoft Services Procurement, tracks deliverable-based projects. This is similar to the time sheet in function and process. The service provider contact uses the progress log to record and submit progress on deliverables. The progress can be logged by the activity that is performed, by the percentage of work that is completed, or by the completion of milestone activities that are defined for the project.

project transaction In PeopleSoft Project Costing, an individual transaction line that represents a cost, time, budget, or other transaction row.

promotion In PeopleSoft Promotions Management, a trade promotion, which is typically funded from trade dollars and used by consumer products manufacturers to increase sales volume.

publishing In PeopleSoft Enterprise Incentive Management, a stage in processing that makes incentive-related results available to participants.

record group A set of logically and functionally related control tables and views. Record groups help enable TableSet sharing, which eliminates redundant data entry. Record groups ensure that TableSet sharing is applied consistently across all related tables and views.

record input VAT flag Abbreviation for *record input value-added tax flag*. Within PeopleSoft Purchasing, Payables, and General Ledger, this flag indicates that you are recording input VAT

	on the transaction. This flag, in conjunction with the record output VAT flag, is used to determine the accounting entries created for a transaction and to determine how a transaction is reported on the VAT return. For all cases within Purchasing and Payables where VAT information is tracked on a transaction, this flag is set to Yes. This flag is not used in PeopleSoft Order Management, Billing, or Receivables, where it is assumed that you are always recording only output VAT, or in PeopleSoft Expenses, where it is assumed that you are always recording only input VAT.
record output VAT flag	Abbreviation for *record output value-added tax flag*. See *record input VAT flag*.
reference data	In PeopleSoft Sales Incentive Management, system objects that represent the sales organization, such as territories, participants, products, customers, channels, and so on.
reference object	In PeopleSoft Enterprise Incentive Management, this dimension-type object further defines the business. Reference objects can have their own hierarchy (for example, product tree, customer tree, industry tree, and geography tree).
reference transaction	In commitment control, a reference transaction is a source transaction that is referenced by a higher-level (and usually later) source transaction, in order to automatically reverse all or part of the referenced transaction's budget-checked amount. This avoids duplicate postings during the sequential entry of the transaction at different commitment levels. For example, the amount of an encumbrance transaction (such as a purchase order) will, when checked and recorded against a budget, cause the system to concurrently reference and relieve all or part of the amount of a corresponding pre-encumbrance transaction, such as a purchase requisition.
regional sourcing	In PeopleSoft Purchasing, provides the infrastructure to maintain, display, and select an appropriate vendor and vendor pricing structure that is based on a regional sourcing model where the multiple ship to locations are grouped. Sourcing may occur at a level higher than the ship to location.
relationship object	In PeopleSoft Enterprise Incentive Management, these objects further define a compensation structure to resolve transactions by establishing associations between compensation objects and business objects.
remote data source data	Data that is extracted from a separate database and migrated into the local database.
REN server	Abbreviation for *real-time event notification server* in PeopleSoft MultiChannel Framework.
requester	In PeopleSoft eSettlements, an individual who requests goods or services and whose ID appears on the various procurement pages that reference purchase orders.
role	Describes how people fit into PeopleSoft Workflow. A role is a class of users who perform the same type of work, such as clerks or managers. Your business rules typically specify what user role needs to do an activity.
role user	A PeopleSoft Workflow user. A person's role user ID serves much the same purpose as a user ID does in other parts of the system. PeopleSoft Workflow uses role user IDs to determine how to route worklist items to users (through an email address, for example) and to track the roles that users play in the workflow. Role users do not need PeopleSoft user IDs.
roll up	In a tree, to roll up is to total sums based on the information hierarchy.
run control	A run control is a type of online page that is used to begin a process, such as the batch processing of a payroll run. Run control pages generally start a program that manipulates data.
run control ID	A unique ID to associate each user with his or her own run control table entries.

run-level context In PeopleSoft Enterprise Incentive Management, associates a particular run (and batch ID) with a period context and plan context. Every plan context that participates in a run has a separate run-level context. Because a run cannot span periods, only one run-level context is associated with each plan context.

search query You use this set of objects to pass a query string and operators to the search engine. The search index returns a set of matching results with keys to the source documents.

section In PeopleSoft Enterprise Incentive Management, a collection of incentive rules that operate on transactions of a specific type. Sections enable plans to be segmented to process logical events in different sections.

security event In commitment control, security events trigger security authorization checking, such as budget entries, transfers, and adjustments; exception overrides and notifications; and inquiries.

serial genealogy In PeopleSoft Manufacturing, the ability to track the composition of a specific, serial-controlled item.

serial in production In PeopleSoft Manufacturing, enables the tracing of serial information for manufactured items. This is maintained in the Item Master record.

session In PeopleSoft Enterprise Learning Management, a single meeting day of an activity (that is, the period of time between start and finish times within a day). The session stores the specific date, location, meeting time, and instructor. Sessions are used for scheduled training.

session template In PeopleSoft Enterprise Learning Management, enables you to set up common activity characteristics that may be reused while scheduling a PeopleSoft Enterprise Learning Management activity—characteristics such as days of the week, start and end times, facility and room assignments, instructors, and equipment. A session pattern template can be attached to an activity that is being scheduled. Attaching a template to an activity causes all of the default template information to populate the activity session pattern.

setup relationship In PeopleSoft Enterprise Incentive Management, a relationship object type that associates a configuration plan with any structure node.

share driver expression In PeopleSoft Business Planning, a named planning method similar to a driver expression, but which you can set up globally for shared use within a single planning application or to be shared between multiple planning applications through PeopleSoft Enterprise Warehouse.

single signon With single signon, users can, after being authenticated by a PeopleSoft application server, access a second PeopleSoft application server without entering a user ID or password.

source transaction In commitment control, any transaction generated in a PeopleSoft or third-party application that is integrated with commitment control and which can be checked against commitment control budgets. For example, a pre-encumbrance, encumbrance, expenditure, recognized revenue, or collected revenue transaction.

SpeedChart A user-defined shorthand key that designates several ChartKeys to be used for voucher entry. Percentages can optionally be related to each ChartKey in a SpeedChart definition.

SpeedType A code representing a combination of ChartField values. SpeedTypes simplify the entry of ChartFields commonly used together.

staging A method of consolidating selected partner offerings with the offerings from the enterprise's other partners.

statutory account Account required by a regulatory authority for recording and reporting financial results. In PeopleSoft, this is equivalent to the Alternate Account (ALTACCT) ChartField.

step In PeopleSoft Sales Incentive Management, a collection of sections in a plan. Each step corresponds to a step in the job run.

storage level In PeopleSoft Inventory, identifies the level of a material storage location. Material storage locations are made up of a business unit, a storage area, and a storage level. You can set up to four storage levels.

subcustomer qualifier A value that groups customers into a division for which you can generate detailed history, aging, events, and profiles.

Summary ChartField You use summary ChartFields to create summary ledgers that roll up detail amounts based on specific detail values or on selected tree nodes. When detail values are summarized using tree nodes, summary ChartFields must be used in the summary ledger data record to accommodate the maximum length of a node name (20 characters).

summary ledger An accounting feature used primarily in allocations, inquiries, and PS/nVision reporting to store combined account balances from detail ledgers. Summary ledgers increase speed and efficiency of reporting by eliminating the need to summarize detail ledger balances each time a report is requested. Instead, detail balances are summarized in a background process according to user-specified criteria and stored on summary ledgers. The summary ledgers are then accessed directly for reporting.

summary time period In PeopleSoft Business Planning, any time period (other than a base time period) that is an aggregate of other time periods, including other summary time periods and base time periods, such as quarter and year total.

summary tree A tree used to roll up accounts for each type of report in summary ledgers. Summary trees enable you to define trees on trees. In a summary tree, the detail values are really nodes on a detail tree or another summary tree (known as the *basis* tree). A summary tree structure specifies the details on which the summary trees are to be built.

syndicate To distribute a production version of the enterprise catalog to partners.

system function In PeopleSoft Receivables, an activity that defines how the system generates accounting entries for the general ledger.

TableSet A means of sharing similar sets of values in control tables, where the actual data values are different but the structure of the tables is the same.

TableSet sharing Shared data that is stored in many tables that are based on the same TableSets. Tables that use TableSet sharing contain the SETID field as an additional key or unique identifier.

target currency The value of the entry currency or currencies converted to a single currency for budget viewing and inquiry purposes.

template A template is HTML code associated with a web page. It defines the layout of the page and also where to get HTML for each part of the page. In PeopleSoft, you use templates to build a page by combining HTML from a number of sources. For a PeopleSoft portal, all templates must be registered in the portal registry, and each content reference must be assigned a template.

territory In PeopleSoft Sales Incentive Management, hierarchical relationships of business objects, including regions, products, customers, industries, and participants.

TimeSpan A relative period, such as year-to-date or current period, that can be used in various PeopleSoft General Ledger functions and reports when a rolling time frame, rather

	than a specific date, is required. TimeSpans can also be used with flexible formulas in PeopleSoft Projects.
trace usage	In PeopleSoft Manufacturing, enables the control of which components will be traced during the manufacturing process. Serial- and lot-controlled components can be traced. This is maintained in the Item Master record.
transaction allocation	In PeopleSoft Enterprise Incentive Management, the process of identifying the owner of a transaction. When a raw transaction from a batch is allocated to a plan context, the transaction is duplicated in the PeopleSoft Enterprise Incentive Management transaction tables.
transaction state	In PeopleSoft Enterprise Incentive Management, a value assigned by an incentive rule to a transaction. Transaction states enable sections to process only transactions that are at a specific stage in system processing. After being successfully processed, transactions may be promoted to the next transaction state and "picked up" by a different section for further processing.
Translate table	A system edit table that stores codes and translate values for the miscellaneous fields in the database that do not warrant individual edit tables of their own.
tree	The graphical hierarchy in PeopleSoft systems that displays the relationship between all accounting units (for example, corporate divisions, projects, reporting groups, account numbers) and determines roll-up hierarchies.
unclaimed transaction	In PeopleSoft Enterprise Incentive Management, a transaction that is not claimed by a node or participant after the allocation process has completed, usually due to missing or incomplete data. Unclaimed transactions may be manually assigned to the appropriate node or participant by a compensation administrator.
universal navigation header	Every PeopleSoft portal includes the universal navigation header, intended to appear at the top of every page as long as the user is signed on to the portal. In addition to providing access to the standard navigation buttons (like Home, Favorites, and signoff) the universal navigation header can also display a welcome message for each user.
user interaction object	In PeopleSoft Sales Incentive Management, used to define the reporting components and reports that a participant can access in his or her context. All Sales Incentive Management user interface objects and reports are registered as user interaction objects. User interaction objects can be linked to a compensation structure node through a compensation relationship object (individually or as groups).
variable	In PeopleSoft Sales Incentive Management, the intermediate results of calculations. Variables hold the calculation results and are then inputs to other calculations. Variables can be plan variables that persist beyond the run of an engine or local variables that exist only during the processing of a section.
VAT exception	Abbreviation for *value-added tax exception.* A temporary or permanent exemption from paying VAT that is granted to an organization. This terms refers to both VAT exoneration and VAT suspension.
VAT exempt	Abbreviation for *value-added tax exempt.* Describes goods and services that are not subject to VAT. Organizations that supply exempt goods or services are unable to recover the related input VAT. This is also referred to as exempt without recovery.
VAT exoneration	Abbreviation for *value-added tax exoneration.* An organization that has been granted a permanent exemption from paying VAT due to the nature of that organization.
VAT suspension	Abbreviation for *value-added tax suspension.* An organization that has been granted a temporary exemption from paying VAT.
warehouse	A PeopleSoft data warehouse that consists of predefined ETL maps, data warehouse tools, and DataMart definitions.

work order In PeopleSoft Services Procurement, enables an enterprise to create resource-based and deliverable-based transactions that specify the basic terms and conditions for hiring a specific service provider. When a service provider is hired, the service provider logs time or progress against the work order.

worksheet A way of presenting data through a PeopleSoft Business Analysis Modeler interface that enables users to do in-depth analysis using pivoting tables, charts, notes, and history information.

worklist The automated to-do list that PeopleSoft Workflow creates. From the worklist, you can directly access the pages you need to perform the next action, and then return to the worklist for another item.

XML schema An XML definition that standardizes the representation of application messages, component interfaces, or business interlinks.

yield by operation In PeopleSoft Manufacturing, the ability to plan the loss of a manufactured item on an operation-by-operation basis.

zero-rated VAT Abbreviation for *zero-rated value-added tax.* A VAT transaction with a VAT code that has a tax percent of zero. Used to track taxable VAT activity where no actual VAT amount is charged. Organizations that supply zero-rated goods and services can still recover the related input VAT. This is also referred to as exempt with recovery.

Index

Numerics/Symbols

A

B

C

D

E

J

R

S

T