



# PeopleTools 8.12 PeopleSoft Business Interlink Runtime Plug-in Programming Guide

**SKU MTBIr8SP1B 1200**

**PeopleBooks Contributors:** Teams from PeopleSoft Product Documentation and Development.

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

Printed in the United States of America.

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

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

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

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

# Contents

## About This PeopleBook

Before You Begin .....	x
Related Documentation .....	x
Documentation on the Internet .....	x
Documentation on CD-ROM .....	xi
Hardcopy Documentation .....	xi
Typographical Conventions and Visual Cues .....	xi
Comments and Suggestions .....	xiii

## Chapter 1

### Introduction to Business Interlinks

What is the Business Interlink Architecture .....	1-1
Running a Business Interlink .....	1-1
Creating a Business Interlink .....	1-3
Example 1: Calculating UPS Shipping Costs .....	1-3
Example 2: Creating a Calculate Cost Business Interlink .....	1-6
Runtime Plug-in Example .....	1-6
List of Common Business Interlink Terms .....	1-7

## Chapter 2

### Setting Up A Business Interlink Runtime Plug-in

Setting up the Development Environment in C++ .....	2-1
Setting up the Development Environment in C++ Under Windows NT .....	2-1
Setting up the Development Environment in C++ Under UNIX .....	2-7
Creating a C++ Template .....	2-10
Setting up the Development Environment in Visual Basic .....	2-11
Setting Up Visual Basic Application Development .....	2-12
Setting up the Development Environment in Java .....	2-16
Setting up the Development Environment in Java Under Windows NT .....	2-17
Setting up the Development Environment in Java Under UNIX .....	2-19
Understanding the Business Interlink SDK Directory Structure .....	2-20
Understanding the Business Interlink SDK Directory Structure: UNIX .....	2-20
Understanding the Business Interlink SDK Directory Structure: Windows NT .....	2-22

## Chapter 3

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

Writing GetVersion for Your Business Interlink Plug-in Class .....	3-1
Writing IsVersionCompatible for Your Business Interlink Plug-in Class .....	3-2
Writing InstantiateDriverInstance for Your Business Interlink Plug-in (C++).....	3-3

## Chapter 4

### Writing the Execution Method for a Runtime Plug-In

Take A Business Interlink Object As Input .....	4-2
Get The Name of Your Business Interlink Object: GetObjName .....	4-2
Get The Configuration Parameters .....	4-3
Get The Input Document: GetInputDocs, ResetCursor .....	4-3
Get The Output Document: GetOutputDocs, Clear .....	4-4
Get The Output Parameter Information: GetOutputParams, size .....	4-4
Loop To Get Every Input Document and the Input Values: GetStatus, GetDoc, GetValue, GetNextDoc .....	4-7
Call The External System .....	4-11
Add The Output Documents and Their Output Values: AddDoc, AddValue, AddNextDoc.....	4-11

## Chapter 5

### Examples of Business Interlink Runtime Plug-in Code

ExecuteTransaction in C++ .....	5-1
ExecuteTransaction in Visual Basic .....	5-10
ExecuteTransaction in Java .....	5-17

## Chapter 6

### Deploying A Business Interlink Runtime Plug-in

Placing on PeopleSoft Application Clients for Testing .....	6-1
Deploying on PeopleSoft Application Servers .....	6-1
Deploying on Web Servers .....	6-1

## Chapter 7

### Testing A Business Interlink Plug-in

Using the Business Interlink Tester: Windows NT .....	7-1
Using the Business Interlink Tester: UNIX .....	7-1
Example of an Input Doc XML file .....	7-2
Writing the Tags in an Input Doc XML File.....	7-3
<Definition>, <Header>, <InterfaceName> Write the definition for this Business Interlink.....	7-4
<Inputs> Write the input values for this Business Interlink.....	7-4

## Chapter 8

### Understanding The Business Interlink Methods

InterfaceObject Class Methods: Accessing Interlink Object Data.....	8-1
ConfigParam .....	8-2
Description .....	8-2
GetDescription, Description.....	8-2
GetInterfaceName, InterfaceName.....	8-3
GetConfigParam, ConfigParam, GetConfigParamCount.....	8-5
GetConfigParams .....	8-6
GetGroup.....	8-8
GetInputDocs .....	8-9
GetInputParams, GetInputParam, GetInputParamCount .....	8-11
GetInterfaceType.....	8-15
GetObjName, ObjName .....	8-16
GetOutputDocs.....	8-18
GetOutputParams, GetOutputParam, GetOutputParamCount .....	8-20
GetRows.....	8-24
InterfaceName .....	8-25
Plug-in Class Methods: Writing a Runtime Plug-in .....	8-25
ExecuteTransaction .....	8-25
ExecuteObjectQuery .....	8-26
ExecuteObjectAdd .....	8-27
ExecuteObjectDelete.....	8-28
ExecuteObjectUpdate.....	8-29
GetVersion, PsIoDriver_GetVer .....	8-30
IsVersionCompatible, PsIoDriver_IsVersionCompatible .....	8-31
Doc Class Methods: Accessing Hierarchical Input/Output Values .....	8-33
AddDoc .....	8-34
AddNextDoc .....	8-39
AddValue, AddValueInt, AddValueFloat, AddValueDouble.....	8-45
Clear .....	8-51
destroy .....	8-52
Empty .....	8-54
GetCount .....	8-55
GetDoc .....	8-60
GetNextDoc.....	8-65
GetPreviousDoc .....	8-70
GetStatus .....	8-75
GetValue .....	8-77
MoveToDoc .....	8-82

ResetCursor .....	8-85
PSIOString methods: Accessing Input/Output Parameter Information (C++ only) .....	8-86
append .....	8-87
c_astr .....	8-88
c_str .....	8-89
find .....	8-90
length.....	8-91
rfind.....	8-92
size.....	8-93
substr .....	8-94
operators +, +=, ==, != .....	8-96
Parameter Lists .....	8-96

## Chapter 9

### Writing the Design-Time Functionality Using Dynamic Catalog Methods

Writing The Design-Time Methods.....	9-2
Write The Plug-in Description: GetDesc .....	9-2
List the Supported Business Interlink Types: IsSupported .....	9-3
For Query and Update Plug-ins, List Relational and Logical Operators: GetCriteriaRelationalOperators, GetCriteriaLogicalOperators .....	9-4
List The Configuration Parameters: GetParameterList .....	9-4
Write the Categories For the Transactions and Classes: GetCategories .....	9-6
Write the Business Class Catalog: GetClassByCategory, GetClassByName .....	9-6
Write the Business Transaction Catalog: GetTransactionByCategory, GetTransactionByName .....	9-8
Writing a Dynamic Catalog: A Short Example.....	9-14
Example of Design Methods (Visual Basic).....	9-16
Example of Design Methods (C++).....	9-28
Dynamic Catalog Class Methods: Writing Design-Time Functionality .....	9-32
FetchNextChunk, PsIoDriver_FetchNextChunk.....	9-33
GetCategories, PsIoDriver_GetCategories.....	9-33
GetClassByName, PsIoDriver_GetClassByName .....	9-35
GetCriteriaRelationalOperators, PsIoDriver_GetCriteriaRelationalOperators .....	9-39
GetCriteriaLogicalOperators, PsIoDriver_GetCriteriaLogicalOperators.....	9-41
GetDesc, PsIoDriver_GetDesc.....	9-42
GetLastErrorMessage.....	9-43
GetClassByCategory, PsIoDriver_GetClassByCategory .....	9-44
GetParameterList, PsIoDriver_GetParameterList .....	9-46
GetTransactionByCategory, PsIoDriver_GetTransactionByCategory.....	9-48
GetTransactionByName, PsIoDriver_GetTransactionByName.....	9-51
IsSupported, PsIoDriver_IsSupported.....	9-53

Transaction/Class Methods: Adding Input/Output Parameters to Transactions, Data Members to Classes .....	9-56
Add .....	9-56
AddInput .....	9-60
AddDataMemberDef .....	9-61
AddOutput .....	9-63
SetClassName .....	9-65
SetTransactionName .....	9-66
push_back .....	9-67

## Chapter 10

### Writing The Execution Method for a Runtime Plug-In (Criteria Data)

Understanding Business Interlink Object Criteria .....	10-2
Understanding the CriteriaRow Structure for Criteria Rows .....	10-3
Understanding the CriteriaNode Structure for Negation and Grouping .....	10-4
Example of CriteriaRowList and CriteriaNodeList .....	10-5
Understanding m_CriteriaGroup .....	10-6
Building A Query String Using The Criteria Methods and Structures .....	10-6
Adding the Outputs to the Query String .....	10-6
Adding the Class Name to the Query String .....	10-7
Adding the Criteria Rows and Groupings to the Query String .....	10-7
Getting The Criteria Rows: GetRows .....	10-8
Getting the Criteria GetGroup() .....	10-8
Getting the Elements From The Criteria Rows .....	10-9

## Index





## ABOUT THIS PEOPLEBOOK

This PeopleBook covers the concepts of Business Interlinks, which allows a PeopleSoft application to integrate with external systems. It discusses how programmers create Business Interlink Runtime Plug-ins.

This PeopleBook is written for programmers who will be integrating external systems with PeopleSoft by writing Business Interlink Runtime Plug-ins. To take full advantage of the information covered in this book, you should also be comfortable using Microsoft® Windows. You should also know a high-level programming language, such as C++ or Visual Basic.

A PeopleSoft application developer will write a Business Interlink XML Design-Time Plug-in. The design of this plug is what you use to create the Business Interlink Runtime Plug-in.



For more information about writing a Business Interlink XML Design-Time Plug-in, see the PeopleSoft Business Interlink Design-Time Plug-in Programming Guide.

---

You, the Business Interlink Software Development Kit Programmer, write a Business Interlink Runtime Plug-in that is based upon the XML Design-Time Plug-in.

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

Setting Up A Business Interlink Runtime Plug-in shows how to set up your development environment for writing a Business Interlink runtime plug-in. It gives examples for Visual C++ and Visual Basic.

Writing the Version Methods for a Business Interlink Runtime Plug-In shows how to write the version methods (GetVersion, IsVersionCompatible) for a Business Interlink runtime plug-in. It gives examples for Visual C++ and Visual Basic.

Writing the Execution Method for a Runtime Plug-In shows how to write a ExecutionTransaction method that accesses the Business Interlink input and output data in a hierarchical structure. It gives examples for Visual C++ and Visual Basic.

Examples of Business Interlink Runtime Plug-in Code lists and discusses complete listings of the ExecuteTransaction method written in Visual C++ and Visual Basic.

Deploying A Business Interlink Runtime Plug-in tells how to deploy runtime plug-ins for others to use.

Testing A Business Interlink Plug-in tells how to run the Business Interlink Tester for UNIX. For Windows NT, it refers to the tester instructions in the PeopleSoft Business Interlink Application Developer Guide.

Understanding The Business Interlink Methods provides syntax and descriptions for the Business Interlink methods that you write and use when you create Business Interlink plug-ins.

Writing the Design-Time Functionality Using Dynamic Catalog Methods shows how to write the dynamic catalog methods. You use these methods when you do not have static catalogs, and therefor you do not use an XML design-time plug-in. It includes an example Visual Basic listing.

Writing The Execution Method for a Runtime Plug-In (Criteria Data) shows how to write the ExecuteObjectQuery method, which is used to write a runtime plug-in that uses criteria data.

## Before You Begin

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

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

## Related Documentation

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



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

---

---

## Documentation on the Internet

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

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

---

## Documentation on CD-ROM

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

---

## Hardcopy Documentation

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

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

### Internet

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

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

### Telephone

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

### Email

Email CPI at [callcenter@conpub.com](mailto:callcenter@conpub.com).

## Typographical Conventions and Visual Cues

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

Please take a moment to review the following typographical cues:


`monospace font`

Indicates PeopleCode.

### **Bold**

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

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

<i>Italics</i>	<p>Indicates a PeopleSoft or other book-length publication. We also use italics for <i>emphasis</i> and to indicate specific field values. When we cite a field value under the page on which it appears, we use this style: <b><i>field value</i></b>.</p> <p>We also use italics when we refer to words as words or letters as letters, as in the following: Enter the number <i>0</i>, not the letter <i>O</i>.</p>
KEY+KEY	Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For ALT+W, hold down the ALT key while you press W.
Jump links	Indicates a jump (also called a link, hyperlink, or hypertext link). Click a jump to move to the jump destination or referenced section.
Cross-references	<p>The phrase For more information indicates where you can find additional documentation on the topic at hand. We include the navigational path to the referenced topic, separated by colons (:). Capitalized titles in <i>italics</i> indicate the title of a PeopleBook; capitalized titles in normal font refer to sections and specific topics within the PeopleBook. Cross-references typically begin with a jump link. Here's an example:</p> <hr/> <p>For more information, see <u>Documentation on CD-ROM</u> in <i>About These PeopleBooks</i>: Related Documentation.</p> <hr/>
<ul style="list-style-type: none"> <li>• Topic list</li> </ul>	Contains jump links to all the topics in the section. Note that these correspond to the heading levels you'll find in the Contents window.
 Name of Page or Dialog Box	Opens a pop-up window that contains the named page or dialog box. Click the icon to display the image. Some screen shots may also appear inline (directly in the text).



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



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



---

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

---

## Comments and Suggestions

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

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

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

[DOC@PEOPLESOFT.COM](mailto:DOC@PEOPLESOFT.COM)

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



## CHAPTER 1

# Introduction to Business Interlinks

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

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

## What is the Business Interlink Architecture

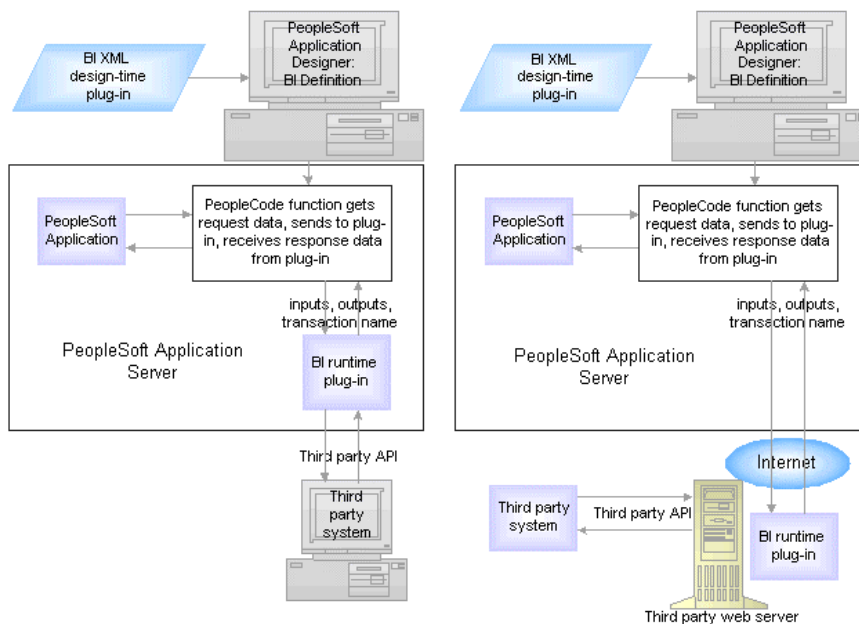
The Business Interlink Architecture provides a plug-in framework for PeopleSoft applications to invoke third party APIs over the Internet. Different vendors support different methods for exposing their APIs, including object technologies such as COM, COBRA, EJB; programming language specific interfaces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

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

---

## Running a Business Interlink

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



## Business Interlink Architecture

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

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



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 and Cost,” the output values could then be displayed upon that page.

---

## Creating a Business Interlink

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

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

## Example 1: Calculating UPS Shipping Costs

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

http://dgiannmon042400/clientservlet.run - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://dgiannmon042400/clientservlet.run Go

Exit Help

**Shipping Time & Cost**  
United Parcel Service

**Ground Time-in-Transit**

\*Origin:

\*Destination:

**QuickCost**

\*Country Origin: United States \*Zip:

\*Country Destination: United States \*Zip:

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

\*Packaging: UPS Express Bo  lbs.

Service Type	Guaranteed By	Commercial Rate

Local intranet

### Shipping Time & Cost Page

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

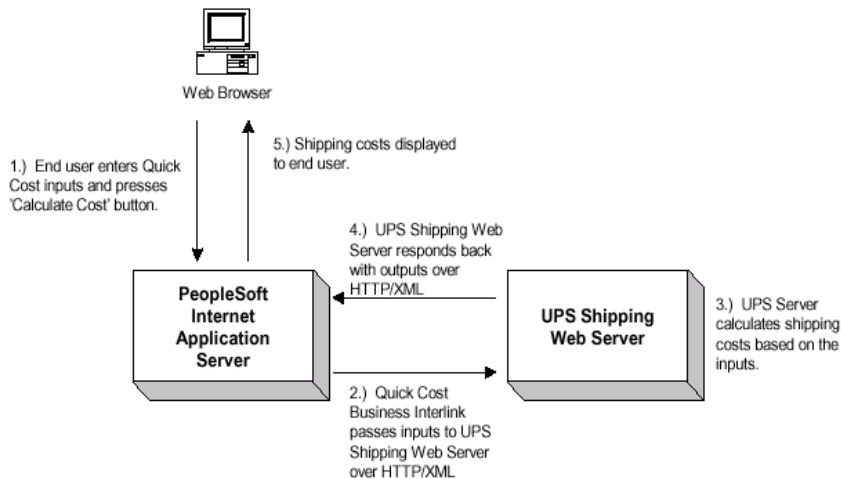
- Ground Time-in-Transit to determine how long it will take to deliver the package.
- Quick Cost to calculate how much it will cost to ship the package based on the shipping service type.

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

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

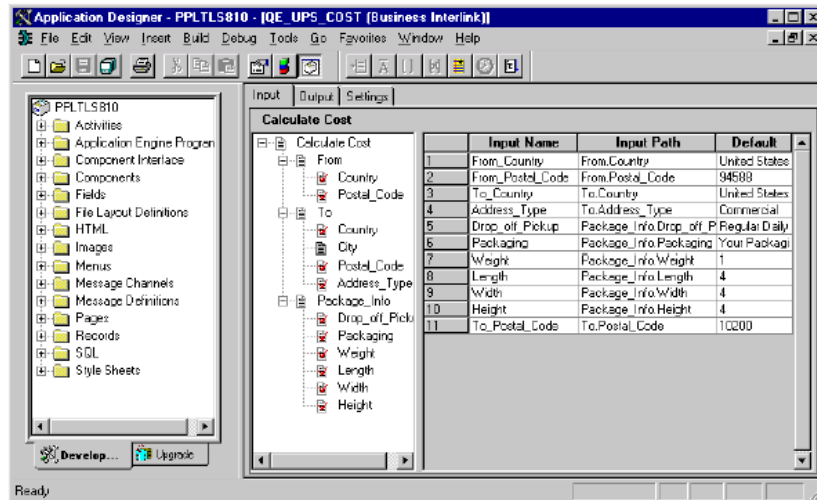
Shipping Time &amp; Cost Page: Calculate Cost Business Interlink

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

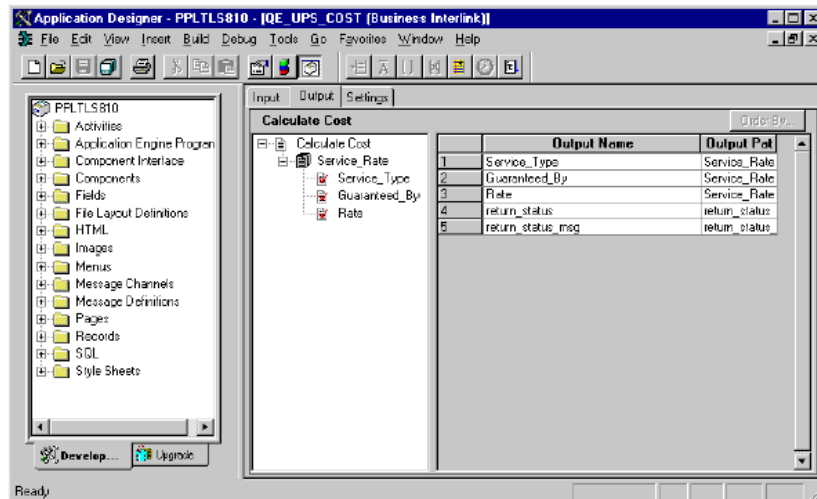


Calculate Cost Business Interlink Processing Flow

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



Calculate Cost Business Interlink Inputs



Calculate Cost Business Interlink Outputs

## Example 2: Creating a Calculate Cost Business Interlink

This section contains code segments that overview a Business Interlink runtime plug-in that calculates shipping costs.

### 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 containing the inputs, outputs, and Business Interlink methods. Since the runtime plug-in encapsulates the external system, PeopleSoft applications can treat all Business Interlinks the same: 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)
    }
}
```

## List of Common Business Interlink Terms

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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





## CHAPTER 2

# Setting Up A Business Interlink Runtime Plug-in

This section shows how to set up your development environment for writing a Business Interlink runtime plug-in. It gives examples for Visual C++ and Visual Basic.

## Setting up the Development Environment in C++

This section contains directions for setting up the development environment in C++ under UNIX and under Windows NT. It also contains a template you can use for a C++ header file.

---

### Setting up the Development Environment in C++ Under Windows NT

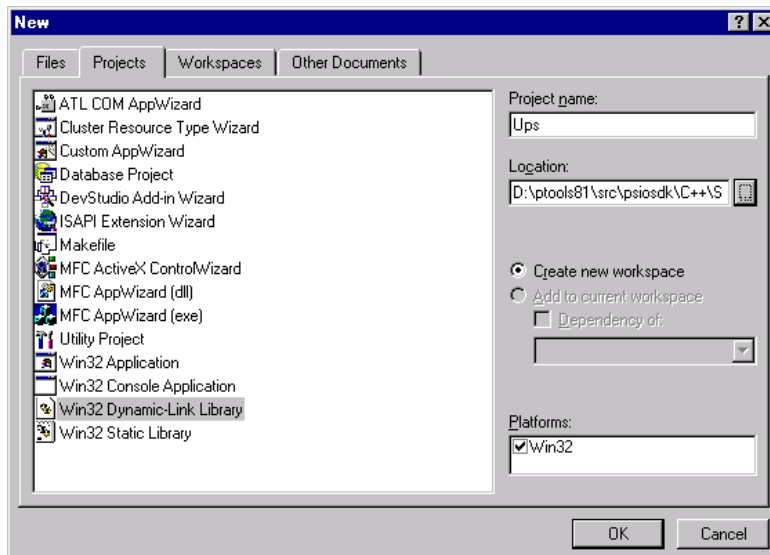
To create the C++ project that you use to write your Business Interlink Plug-in under Windows NT

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.

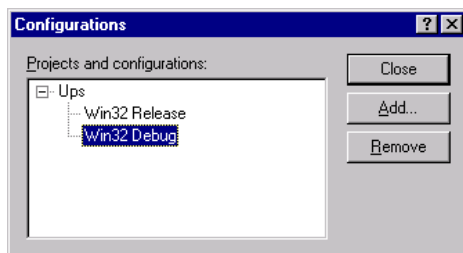
In the project name type the name of the project. In this example, the name of the project is Ups.

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



New Project Dialog Box, Projects Tab

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



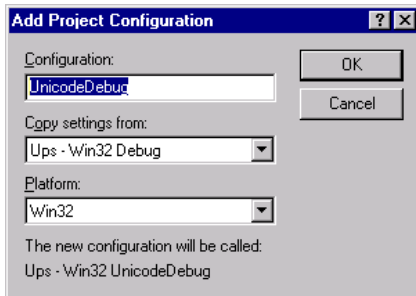
Configurations Dialog Box

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:

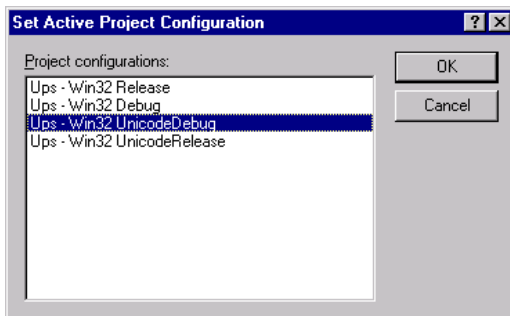
In the Configuration edit box, change the text from “Debug” to “UnicodeDebug”. This is set because, by default, the project is not Unicode enabled, so Unicode must be enabled here.

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



Add Project Configuration dialog box

6. Repeat steps 4 and 5, except in step 5:
  - Within the “Copy settings from” drop down box in the Add Project Configuration dialog box, select the Win32 Release configuration as the template.
  - Within the Configuration edit box in the Add Project Configuration Dialog Box, change the text to “Unicode Release”.
  - Click the Close button.
7. Select Build:Set Active Configuration. The Set Active Project Configuration dialog box appears. Select Win32 UnicodeDebug and click OK.



Set Active Project Configuration

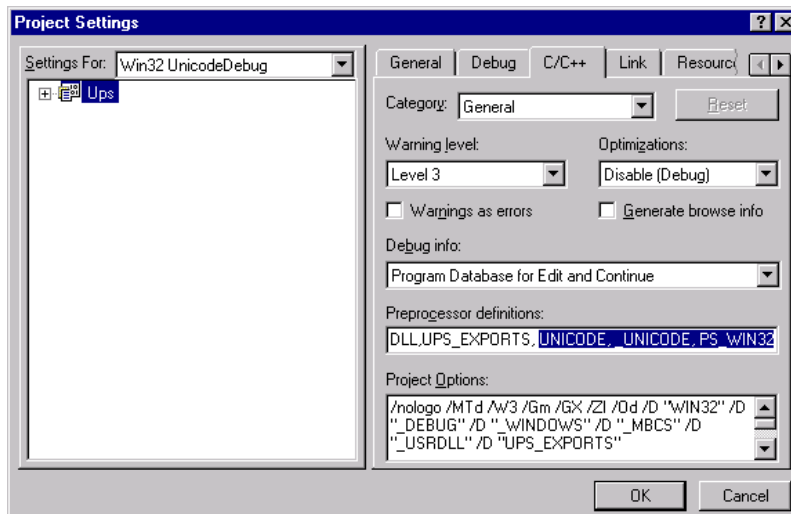
8. Select Project:Settings. The Project Setting dialog box appears.

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.

9. Click the C/C++ tab. 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.

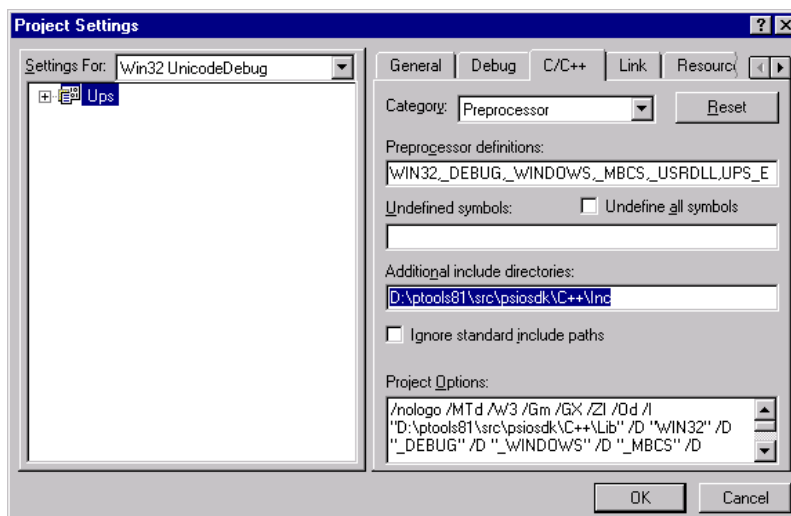


Project Settings dialog box, C/C++ tab, General category

10. From the Category drop down box, choose the Preprocessor category. 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.



Project Settings dialog box, C/C++ tab, Preprocessor category

11. Click on the Link tab. 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:

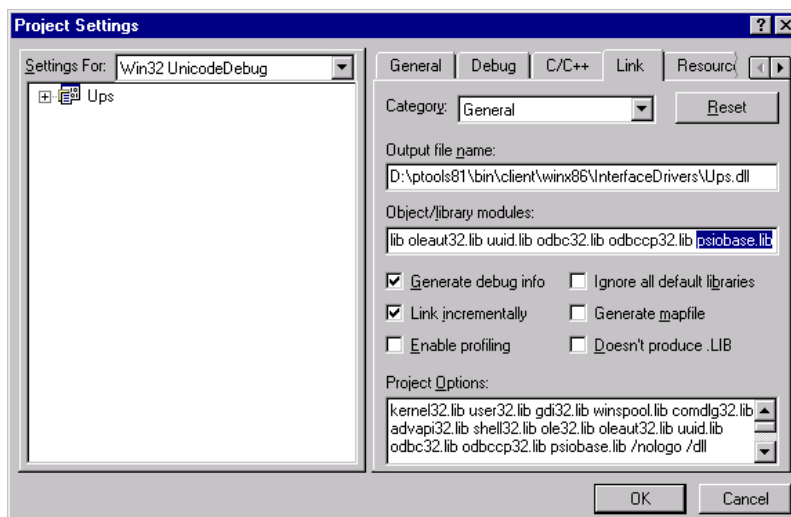
```
<PS_HOME>\bin\client\winx86\InterfaceDrivers\output.dll
```

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 runtime is *fcarrier.dll*, the design-time should be *fcarrier.xml*.



For more information about the XML design-time plug-in, see the *PeopleSoft Business Interlink Design-Time Plug-in Programming Guide*, Writing an XML Design-Time Plug-In.

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.

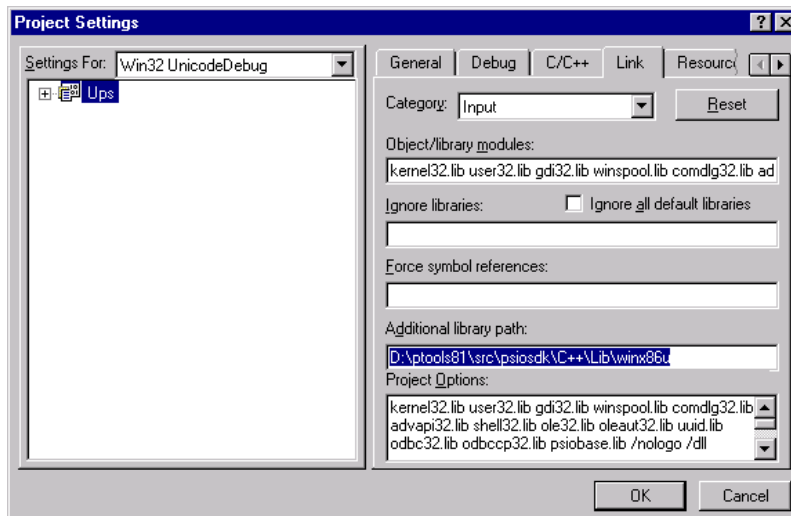


Project Settings dialog box, Link tab, General category

12. From the Category drop down box, select Input. 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.



Project Settings dialog box, Link tab, Input category

13. Press 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 (in this case, it will likely be Ups).

14. 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
```

You can use the following code as a template for the header file for 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. In this case, the name of your class is *CUps*; you will use whatever name fits your project.
- 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 CUps : public DLLBaseDriver
{
public:
    // Constructor for your class.
    CUps() : IntObj(0) {}


    // Declare the virtual methods
    virtual const TCHAR* GetVersion() const;
    virtual BOOL IsVersionCompatible(const TCHAR* version);
    virtual BIOCEXECSTATUS
        ExecuteTransaction(InterfaceObject * IntObj,
            const int nBatchMode);


    // Create the InterfaceObject pointer.
public:
    InterfaceObject* IntObj;
};

```

---

## Setting up the Development Environment in C++ Under UNIX

In order to write a Business Interlink runtime plug-in using C++, you set up the environment for Business Interlinks on the UNIX system.

To create the C++ project that you use to write your Business Interlink Plug-in under UNIX

1. Change directories to the <PS\_HOME> directory and run the following command:

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

2. If you wish, you can test the setup by running the Business Interlink tester on the files in the simple directory. If the tester works, the setup installed properly.



For more information on running the Business Interlink Tester under UNIX, see Testing A Business Interlink Plug-in.

---

3. Create the directory in which you will create your plug-in. (A recommended location is <PS\_HOME>/sdk/psinterlinks/src/C++/samples.) Within that directory, create a directory named `unix`.

```
mkdir yourplugindir
```

```
cd yourplugindir
```

```
mkdir unix
```

where *yourplugindir* is the name of the directory in which you will create your plug-in.

4. Copy the files named `makefile` and `makeunix.mak` from the `newplugin/unix` directory to *yourplugindir/unix*, where *yourplugindir* is the name of the directory that you created in the previous step.

```
cd ..
```

```
cp newplugin/unix/* yourplugindir/unix/*
```

5. Edit the file named `makefile` in the *yourplugindir/UNIX* directory.

Change the following line:

```
name=                newplugin
```

replacing `newplugin`.

```
name=                yourplugindir
```

where *yourplugindir* is the name of the directory you created earlier in which you will create your plug-in.

Change the following lines, replacing `newfile1` with the name of any `.cpp` file that you will have within the *yourplugindir* directory. You must copy this code and make a similar change for every `.cpp` file that you have within the *yourplugindir* directory.

```
newfile1.o : $(srcdir)/newfile1.cpp
```

```
$(CC) $(dbgflags) $(cmptopts) $(typedefine) $(defines) $(unicodedefs)
$(libincludes) $(includes) $(tuxincludes) $(srcdir)/newfile1.cpp -E > newfile1.x
```



```
wconv $(unicodedefs) newfile1.x newfile1.i

$(CC) $(cmpopts) $(dbgflags) $(typedefine) $(defines) $(unicodedefs)
$(libincludes) $(includes) $(tuxincludes) newfile1.i -c

rm newfile1.x
```

After replacing newfile1 with yourplugin:

```
yourplugin.o : $(srcdir)/yourplugin.cpp

$(CC) $(dbgflags) $(cmpopts) $(typedefine) $(defines) $(unicodedefs)
$(libincludes) $(includes) $(tuxincludes) $(srcdir)/yourplugin.cpp -E >
yourplugin.x

wconv $(unicodedefs) yourplugin.x yourplugin.i

$(CC) $(cmpopts) $(dbgflags) $(typedefine) $(defines) $(unicodedefs)
$(libincludes) $(includes) $(tuxincludes) yourplugin.i -c

rm yourplugin.x
```

6. Copy the file named `psiodrvr.cpp` into the *yourplugindir* directory. Change the following lines to match the names of your plug-in (*yourplugin*):

```
#include          "windows.h"

#include          "iodrvr.h"

#include          "yourplugin.h"

#define EXPORT_PSTYP __declspec(dllexport)

static yourplugin baseDriver;
```

7. Create your Business Interlink Runtime plug-in. Store the `.cpp` and `.h` files within the *yourplugindir* directory. You can copy, rename, and use the `.cpp` and `.h` files contained in the simple directory as a template.
8. To compile and link your runtime plug-in, run the make command:

```
make yourplugin
```

where *yourplugin* is the name of your runtime plug-in.

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`.



For more information about the XML design-time plug-in, see the ***PeopleSoft Business Interlink Design-Time Plug-in Programming Guide***, Writing an XML Design-Time Plug-In.

---

---

## 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. In this case, the name of your class is *CUps*; you will use whatever name fits your project.
- 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

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, if you have installed PeopleTools on your D: drive in a directory names ptool81, you might use the following command 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
```

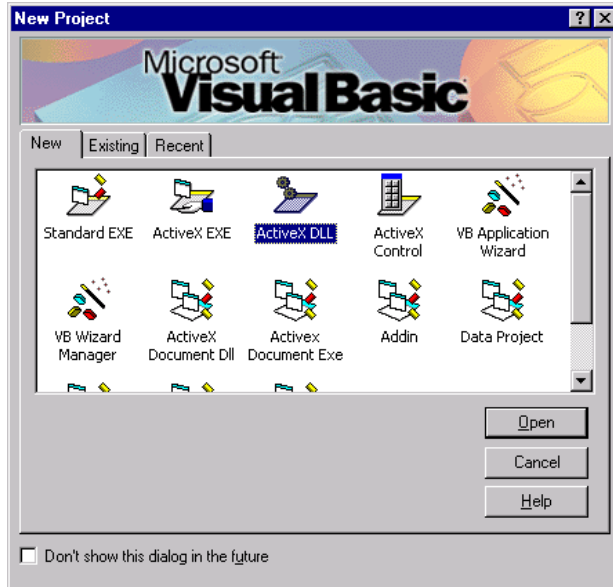
```
C:\winnt\system32\regsvr32.exe PsIntObj.DLL
```

---

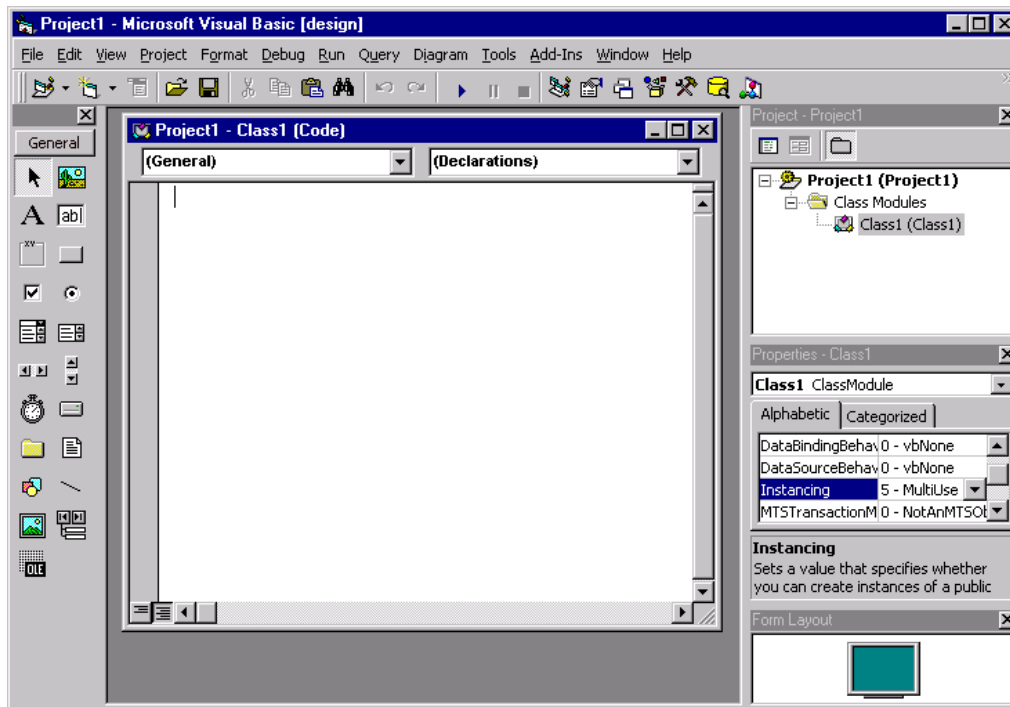
## Setting Up Visual Basic Application Development

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.

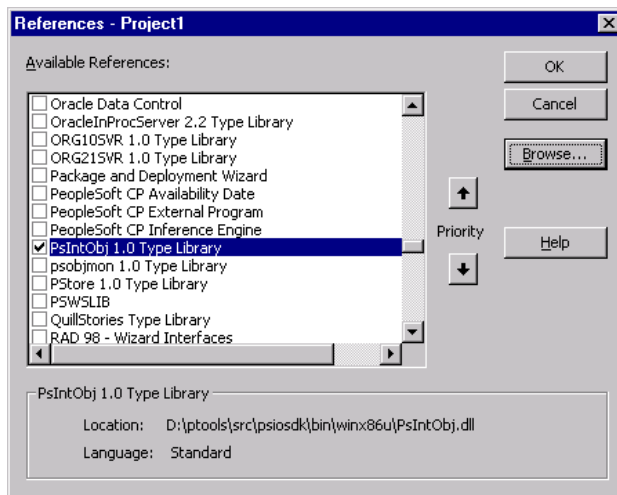


Selecting ActiveX.DLL



A New Project Window

- From the main menu, select Project:References. The References window appears.



Available References Window

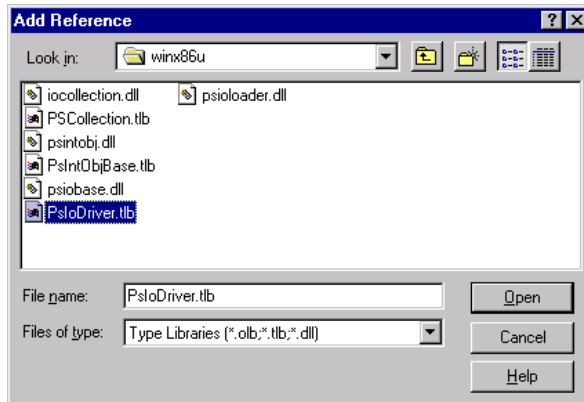
- From References option menu check out the following DLLs and TLB:

PsIoDriver.tlb. Press 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

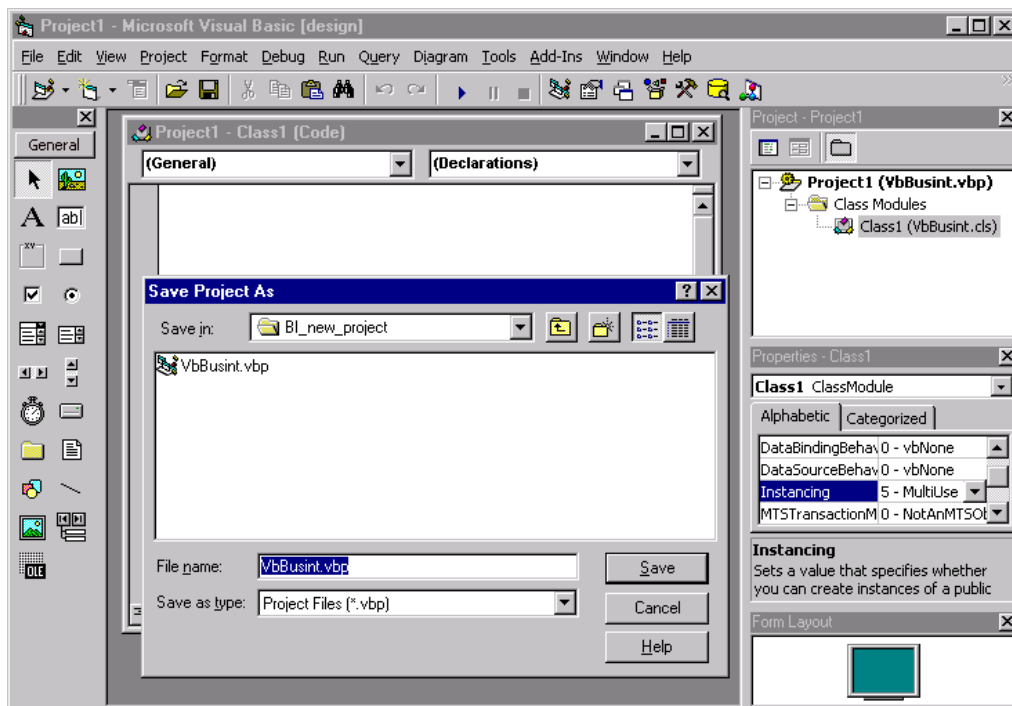
IoCollection.DLL and PsIntObj.DLL. You can either use the References window and check PsIntObj 1.0 Type Library and IoCollection 1.0 Type Library, or you can use the Add References window to navigate to the directory containing IoCollection.DLL and PsIntObj.DLL, and select them. The directory is located at:

<PS\_HOME>\bin\client\winx86



Add References Window

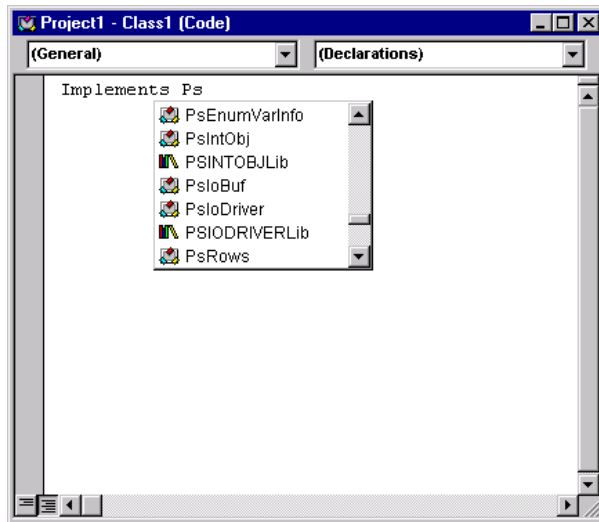
- From main menu select Save Project. Navigate to the directory where you want to save your project, and save it.



Saving The Project

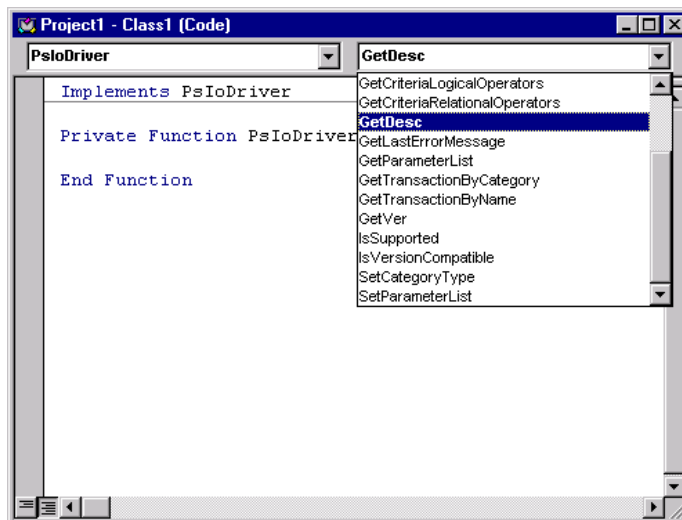
- Type in “Implements PsIoDriver” in your Class (code) screen, then select PsIoDriver from

the Class category pop-up list box. .



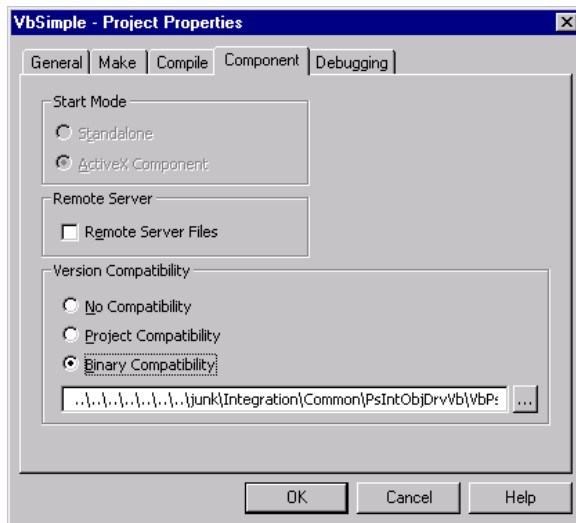
Implements Window

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.



Select Functions Listbox

8. Set the level of version compatibility of your VB project to Binary. Open your Visual Basic project, and from the Project menu, select *yourproject* Properties, where *yourproject* is the name of your project. In the picture below, the project is names VbSimple. Then click the Binary Compatability radio button and click OK.



Select Binary Compatability

9. Implement the functions provided by PsIoDriver that you will need for your project.



For more information on the functions that you implement, refer to Writing the Version Methods for a Business Interlink Runtime Plug-In and Writing the Execution Method for a Runtime Plug-In.

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.



For more information on writing methods instead of an XML design-time plug-in, see Writing the Design-Time Functionality Using Dynamic Catalog 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.



For more information about the XML design-time plug-in, see the *PeopleSoft Business Interlink Design-Time Plug-in Programming Guide*, Writing an XML Design-Time Plug-In.

## Setting up the Development Environment in Java

This section contains directions for setting up the development environment in Java under UNIX and under Windows NT.



## Setting up the Development Environment in Java Under Windows NT

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 1.2.2 and Java Runtime Environment 1.2.2.

Java Development Kit version 1.2.2 is located at:

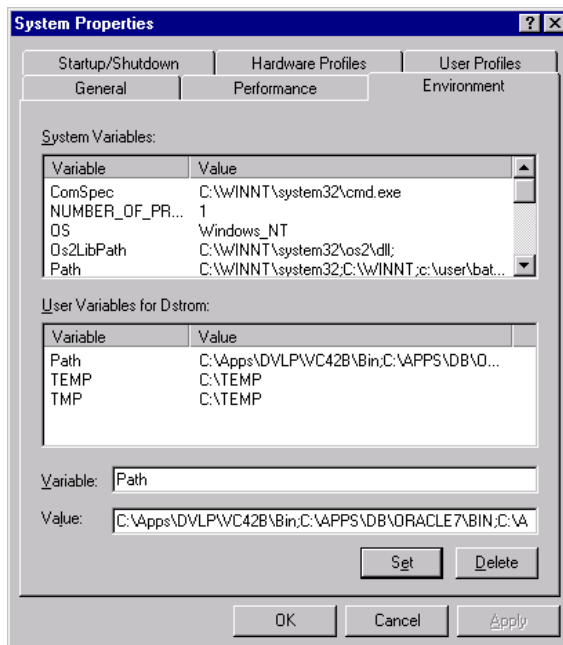
<http://java.sun.com/products/jdk/1.2/>

Java Runtime Environment is located at:

<http://java.sun.com/products/jdk/1.2/jre/index.html>

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

C:\<PS\_HOME>\class\psinterlinks.jar



System Properties panel, Environment Tab

3. 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:

C:\Program Files\Javasoftware\Jre\1.2\bin

C:\Program Files\Javasoftware\Jre\1.2\bin\Classic

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 \appsrv\<>name of application server domain>):

```
C:\Program Files\JavaSoft\Jre\1.2\bin\Classic
```

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\yourjavaplugin.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.



For more information about the XML design-time plug-in, see the ***PeopleSoft Business Interlink Design-Time Plug-in Programming Guide***, Writing an XML Design-Time Plug-In.

---

---

## Setting up the Development Environment in Java Under UNIX

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

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/jdk/1.2/`

Java Runtime Environment is located at:

`http://java.sun.com/products/jdk/1.2/jre/index.html`

2. Copy the libpsjavaproxy file into the directory where you are developing your Java plug-in. 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, libpsjavaproxy.sl
3. Write your Java runtime plug-in. The `<PS_Home>/appserv/classes` 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`



For more information on writing the runtime plug-in, see Writing the Version Methods for a Business Interlink Runtime Plug-In and Writing the Execution Method for a Runtime Plug-In.

---

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.



For more information about the XML design-time plug-in, see the *PeopleSoft Business Interlink Design-Time Plug-in Programming Guide*, Writing an XML Design-Time Plug-In.

## Understanding the Business Interlink SDK Directory Structure

This section describes the Business Interlink directory structures for Windows NT and UNIX.

### Understanding the Business Interlink SDK Directory Structure: UNIX

The Business Interlink directory structure on your UNIX 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/psinterlinks/
- bin, which contains the file bitest, which is the executable for the UNIX Business Interlink tester. bin also contains the file simple.xml, which is the XML design-time plug-in for the simple runtime plug-in, and the files addnumber.xml, catstr.xml, and machine.xml, which are Input Doc xml files for the simple plug-ins.
- 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.
  - `samples` 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)

libpsjavaproxy

sdk

psinterlinks

bin

addnumber.xml

catstr.xml

machine.xml

simple.xml

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
```

---

## Understanding 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 3

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

The version information consists of writing the following methods:

- **GetVersion** or **GetVer**. This method provides the version number for your Business Interlink Plug-in.
- **IsVersionCompatible**. This method tells whether or not your Business Interlink Plug-in is compatible with previous versions of your Business Interlink Plug-in.
- **InstantiateDriverInstance**. This method creates an instance of 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 * CUps::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 UPS 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 run-time plug-in will be compatible with versions 8.0 and 8.1. Then you could write IsVersionCompatible 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();
```



# Writing the Execution Method for a Runtime Plug-In

The execution method performs the bulk of the work for a runtime plugin. 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.



For more information about writing `ExecuteObjectQuery`, `ExecuteObjectUpdate`, or `ExecuteObjectDelete`, which use the Criteria methods, see [Writing The Execution Method for a Runtime Plug-In \(Criteria Data\)](#).

---



For more information about the methods that are mentioned here, or any other Business Interlink methods that you can use when writing a runtime plug-in, refer to [Understanding The Business Interlink Methods](#).

---

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.

---



**For a listing** of the `ExecuteTransaction` written in C++, see [ExecuteTransaction in C++](#).

---




---

**For a listing** of the ExecuteTransaction written in Visual Basic, see ExecuteTransaction in Visual Basic.

---

## 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 CUps::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)
```




---

The nBatchMode and lBatchMode parameters are not currently used.

---

## Get The Name of Your Business Interlink Object: GetObjName

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.




---

For more information about writing an XML design-time plug, see *PeopleSoft Business Interlink Design-Time Plug-in Programming Guide*, Writing an 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"))
```



For more information about the **GetObjName** method, see [GetObjName](#), [ObjName](#).

---

## Get The Configuration Parameters

In the case of the Freight Carrier plug-in, there are no configuration parameters that are used in `ExecuteTransaction`.



For more information about getting configuration parameters, see [GetConfigParams](#).

---

## Get The Input Document: **GetInputDocs**, **ResetCursor**

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
```



For more information about the **GetInputDocs** method and the **ResetCursor** method, see **GetInputDocs** and **ResetCursor**.

## Get The Output Document: **GetOutputDocs**, **Clear**

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();
```



For more information about the **GetOutputDocs** method and the **Clear** method, see **GetOutputDocs** and **Clear**.

## Get The Output Parameter Information: **GetOutputParams**, **size**

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.



**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.**”.



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++:



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:




---

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++)

```



For more information about the parameter list structure used by VarInfo, PsEnumVarInfo, and VARINFOLIST, see Parameter Lists.

---



For more information about the **GetOutputParams** and **size** methods, refer to GetOutputParams, GetOutputParam, GetOutputParamCount and size.

---

## 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.



You could also use the **GetCount** method to loop through 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")

End While
```

```

' 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();

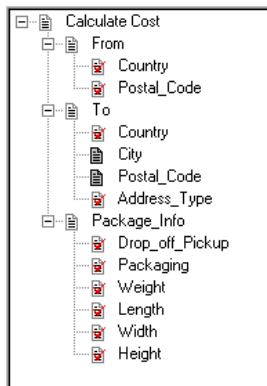
    // Call the external system using these input values, insert the
    // output values into the output document (code not shown here)

}

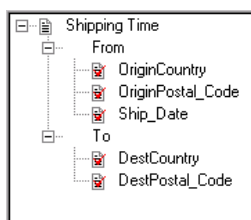
docsIn.destroy();

```

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 figures below show the documents that **GetDoc** gets in this example.



Example Input Document: Calculate Cost



Example Input Document: Shipping Time



For more information about GetCount, GetDoc, GetNextDoc, GetStatus, and GetValue, see GetCount, GetDoc, GetNextDoc, GetStatus, and GetValue.

## 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.

## Add The Output Documents and Their Output Values: AddDoc, AddValue, AddNextDoc

Loop to add each output document (set of output parameters) and add the output values to the output document.



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.



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(upsinfo.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();

    }

}

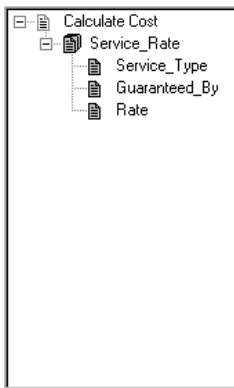
docsOut.AddValue(_T("return_status"), _T("0"));

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:



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

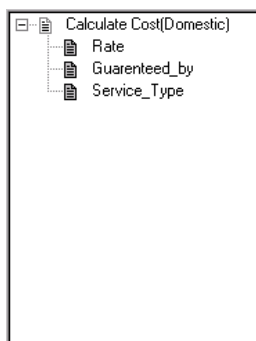
    CNT = CNT + 1

Wend

objInputDocs.GetNextDoc

```

The figure below shows the output document for the Visual Basic example. It contains three output parameters: Rate, Guarenteed\_by, and Service\_Type.



Example PsBIDocs Output Document for Visual Basic  
In Visual Basic:



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

---



For more information about **Empty**, **AddDoc**, **AddValue**, and **AddNextDoc**, see **Empty**, **AddDoc**, **AddValue**, **AddValueInt**, **AddValueFloat**, **AddValueDouble**, and **AddNextDoc**.

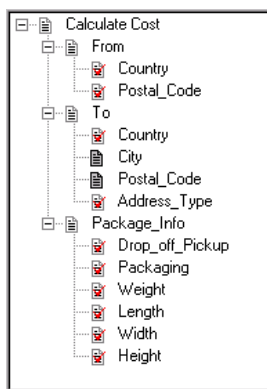
---

## CHAPTER 5

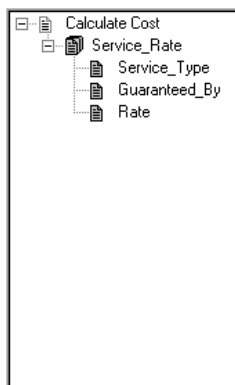
# Examples of Business Interlink Runtime Plug-in Code

### ExecuteTransaction in C++

For reference, here is the structure of the input and output documents for the Calculate Cost Business Interlink Object.



Calculate Cost Business Interlink Object Inputs



Calculate Cost Business Interlink Object Outputs

The following code shows the ExecuteTransaction for a UPS 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 CUps::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_UPS_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.ups.com/using/services/rave/qcost.cgi"),

        (TCHAR*)(LPCTSTR)strPostData,

        strPostData.GetLength(),

        (TCHAR *)strFileName.c_str()))

    {

        CUpsInfo upsinfo(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(upsinfo.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 (upsinfo.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;

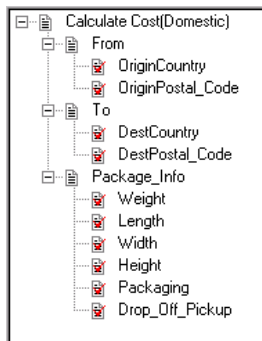
```

```
}

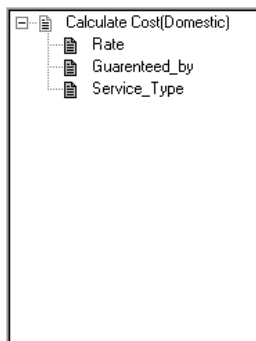
```

## ExecuteTransaction in Visual Basic

For reference, here is the structure of the input and output documents for the Calculate Cost(Domestic) Business Interlink Object.



### Calculate Cost(Domestic) Business Interlink Object Inputs



### 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

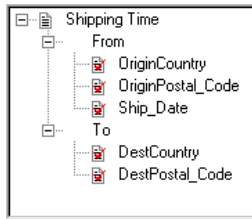

End If


ExecuteTransaction = ENUM_INTOBJ_SUCCEED
'

End Function
```

## ExecuteTransaction in Java

For reference, here is the structure of the input and output documents for the Shipping Time Business Interlink Object.



### Shipping Time Business Interlink Object Inputs



### 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 wrapped 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 6

# Deploying A Business Interlink Runtime Plug-in

After you have written and tested your Business Interlink runtime plug-ins, you can deploy them for others to use.

## Placing on PeopleSoft Application Clients for Testing

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.



---

For more information on the Business Interlink tester, see *PeopleSoft Business Interlink Application Developer Guide*, Business Interlink Tester Page: Testing Your Business Interlink Definition.

---

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) is located in the following location:

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

Where <PS\_Home> is the directory where PeopleTools is installed.

## 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) 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:

For Microsoft IIS, the default location is:

```
c:\inetpub\wwwroot\PsInterlinks\interfacedrivers
```

For Apache Web Server, the default location is:

```
c:\program files\apache jserv 1.1\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 more information about installation, see the Installing External Installation chapter in *PeopleSoft 8 Installation and Administration*.

---

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.



For more information about the XML design-time plug-in, see the *PeopleSoft Business Interlink Design-Time Plug-in Programming Guide*, Writing an XML Design-Time Plug-In.

---

## CHAPTER 7

# Testing A Business Interlink Plug-in

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.



For more information about the XML design-time plug-in, see PeopleSoft Business Interlink Design-Time Plug-in Programming Guide.

---



For more information on running the Business Interlink Tester under Windows NT, see Business Interlink Tester Page: Testing Your Business Interlink Definition.

---

## Using the Business Interlink Tester: UNIX

Once you have set up Business Interlinks on UNIX, you can test your Business Interlink runtime plug-ins using the Business Interlink tester for UNIX. 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:

1. Create an Input Doc XML file.



For more information on creating an Input Doc XML file, see Example of an Input Doc XML file and Writing the Tags in 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.



For more information about the XML design-time plug-in, see the PeopleSoft Business Interlink Design-Time Plug-in Programming Guide.

---

3. Within the <PS\_HOME>/bin directory, run the following command to run the Business Interlink Tester for UNIX.

```
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 untared with the Setting up Business Interlinks on UNIX 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 untared with the Setting up Business Interlinks on UNIX 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*).

---

## Example of an Input Doc XML file

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_list>

    </transaction>

  </category>

</trans_catalog>
```

```

        <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>

```

---

## Writing the Tags in an Input Doc XML File

The main tag for the Business Interlink XML design-time plug-in is **<Business\_Interlink>**. It contains the **<Definition>** and **<Inputs>** tags.

**<Definition>, <Header>, <InterfaceName> Write the definition for this Business Interlink**

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>
```

**<Inputs> Write the input values for this Business Interlink**

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.

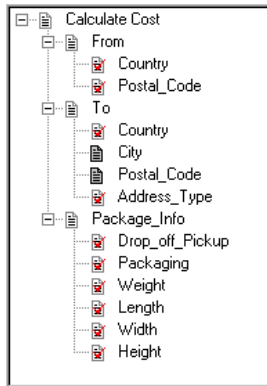
```
<Inputs>

  <number_1>3</number_1>

  <number_2>23</number_2>

</Inputs>
```

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>
```



## CHAPTER 8

# Understanding The Business Interlink Methods

**InterfaceObject Class Methods:** Use these methods when you write your execution method (or methods) for your Interlink class.

**Plug-in Class Methods:** Runtime. Write these methods when you are creating a runtime plug-in. The execution methods are in this class.

**CBIDocs methods:** Use these methods to access the input and output values of a Business Interlink Object when the input and output is hierarchical.

**PSIOString methods:** Use these methods 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: Accessing Interlink Object Data

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 table shows which InterfaceObject methods you use to retrieve Business Interlink Object information.

<b><i>To retrieve:</i></b>	<b><i>Use this method</i></b>
A text description of the Business Interlink Object	GetDescription(), GetDesc
The name of the Business Interlink Definition.	GetInterfaceName()
The value of one configuration parameter.	GetConfigParam(strParamName)
The list of configuration parameters for this Business Interlink Object.	GetConfigParams()
The input parameter information for this Business Interlink Object.	GetInputParams()
The input table containing the input values for	GetInputTable()

this Business Interlink Object.	
The type of this Business Interlink Object.	GetInterfaceType()
The name of this Business Interlink Object.	GetObjName()
The output parameter information for this Business Interlink Object.	GetOutputParams()
The output table into which you will insert output values for this Business Interlink Object.	GetOutputTable()

## ConfigParam

See GetConfigParam, ConfigParam, GetConfigParamCount.

## Description

See GetDescription, Description.

## GetDescription, Description

### Syntax: C++

```
PSIOString& GetDescription()
```

### Syntax: Visual Basic

```
string Description
```

### Syntax: Java

```
String GetDescription()
```

## Class

C++: InterfaceObject

Visual Basic: IpsIntObj

Java: PSJInterfaceObject

## 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.

## Parameters

None.

## Return Value

<b>Value</b>	<b>Meaning</b>
C++: PSIOString	A string containing the description for this Business Interlink Plug-in.
Visual Basic: string	
Java: String	

## 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>UPS services</description>
  <version>1</version>
</general_info>
```

the description returned would be “UPS services”.

## GetInterfaceName, InterfaceName

### Syntax: C++

```
PSIOString& GetInterfaceName()
```

### Syntax: Visual Basic

```
string InterfaceName
```

**Syntax: Java**

```
String GetInterfaceName()
```

**Class**

C++: InterfaceObject

Visual Basic: IpsIntObj

Java: PSJInterfaceObject

**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.

**Parameters**

None.

**Return Value**

<i><b>Value</b></i>	<i><b>Meaning</b></i>
C++: PSIOString	A string containing the Business Interlink Definition name.
Visual Basic, Java: string	

**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_UPS_COST);
```

The following code within a Business Interlink Plug-in would fill the IntName variable with the Business Interlink Definition name of "QE\_UPS\_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&);
```

### Syntax: Visual Basic

```
string ConfigParam
```

### Syntax: Java

```
PSJVarInfo GetConfigParam(int index)
```

```
int GetConfigParamCount()
```

### Class

C++: InterfaceObject

Visual Basic: IPsIntObj

Java: PSJInterfaceObject

### 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.

### Parameters

strParamName: Input. A PSIOString variable containing the name of the configuration parameter. This name is set either in the XML design-time plug-in or by the

### Return Value

<b>Value</b>	<b>Meaning</b>
PSIOString&: C++	A string containing the value of the configuration parameter named in the strParamName input parameter.
string: Visual Basic, Java	

## 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()
```

## Syntax: Visual Basic

```
GetConfigParams() As IPsEnumVarInfo
```

## Class

C++: InterfaceObject

Visual Basic: IPsIntObj

## 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.

## Parameters

None.

## Return Value

A list of type VARINFOLIST in C++, and PSEnumVarInfo in Visual Basic.



For more information about VARINFOLIST and PSEnumVarInfo, see Parameter Lists.

---

## 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 UniCode data.
- `c_astr` if your software system uses ASCII data.

For example, to get the names of the configuration parameters using UniCode data, use the `c_str()` method and the `m_strName`

```
char *config_name1 = (char *)varList[0]->m_strName.c_str();
```

```
char *config_name2 = (char *)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()
```

### Class

C++: CriteriaGroup

### Description

The method **GetGroup** gets a CriteriaNodeList structure. This structure will contain the grouping for the criteria rows for a query or update.



For more information about criteria rows and the CriteriaNodeList structure, see Understanding Business Interlink Object Criteria.

---



## Parameters

None.

## Return Value

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*)
```

### Syntax: Visual Basic

```
GetInputDocs() As IPsBIDocs
```

### Syntax: Java

```
PSJBIDocs GetInputDocs(String)
```

## Class

C++: InterfaceObject

Visual Basic: IpsIntObj

Java: PSJInterfaceObject

## 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.

## Parameters

None. (C++ and Java pass in an empty string.)

## Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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();
```

## Related Topics

Doc Class Methods: Accessing Hierarchical Input/Output Values

## GetInputParams, GetInputParam, GetInputParamCount

### Syntax: C++

```
VARINFOLIST& GetInputParams()
```

### Syntax: Visual Basic

```
GetInputParams() As IPsEnumVarInfo
```

### Syntax: Java

```
PSJVarInfo GetInputParam(int index)
```

```
int GetInputParamCount()
```

## Class

C++: InterfaceObject

Visual Basic: IPsIntObj

Java: PSJInterfaceObject

## 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.

## Parameters

None for `GetInputParams` or `GetInputParamCount`.

For `GetInputParam`:

## Parameters

index	The location of the input parameter in the parameter list. Its value can be 0 to (number of parameters - 1).
-------	---

## Return Value

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.



For more information about `VARINFOLIST` and `IPSEnumVarInfo`, see [Parameter Lists](#).

## 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"/>

  </output_list>

</transaction>
```

```

        <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()
```

### Syntax: Visual Basic

```
string InterfaceType
```

### Syntax: Java

```
int GetInterfaceType()
```

### Class

C++: InterfaceObject

Visual Basic: IPsIntObj

Java: PSJInterfaceObject

### 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.

### Parameters

None.

### Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
EIOCINTERFACESUPPORTED	The type of the Business Interlink Object. This can be any of the following:
	IOC_TRANSACTION (transaction)
	IOC_OBJADD (object add)
	IOC_OBJQUERY (object query)
	IOC_OBJUPDATE (object update)
	IOC_OBJDELETE (object delete)

## 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;

    case IOC_OBJQUERY:
        eRt = ExecuteObjectQuery(IntObj, nBatchMode);
        break;

    case IOC_OBJADD:
        eRt = ExecuteObjectAdd(IntObj, nBatchMode);
        break;

    case IOC_OBJDELETE:
        eRt = ExecuteObjectDelete(IntObj, nBatchMode);
        break;

    case IOC_OBJUPDATE:
        eRt = ExecuteObjectUpdate(IntObj, nBatchMode);
        break;

    default:
        eRt = INTOBJ_FAILED;
}
```

## GetObjName, ObjName

### Syntax: C++

```
PSIOString& GetObjName()
```



**Syntax: Visual Basic**

```
string ObjName
```

**Syntax: Java**

```
int GetObjName()
```

**Class**

C++: InterfaceObject

Visual Basic: IPsIntObj

Java: PSJInterfaceObject

**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 created in the XML design-time plug-in.

**Parameters**

None.

**Return Value**

<i><b>Value</b></i>	<i><b>Meaning</b></i>
C++: PSIOString&	A string containing the Business Interlink Object name.
Visual Basic: string	
Java: String	

**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*)
```

### Syntax: Visual Basic

```
GetOutputDocs() As IPsBIDocs
```

### Syntax: Java

```
PSJBIDocs GetOutputDocs(String)
```

## Class

C++: InterfaceObject

Visual Basic: IPsIntObj

Java: PSJInterfaceObject

## 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.

## Parameters

None. (C++ and Java pass in an empty string.)

## Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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();
```

## Related Topics

Doc Class Methods: Accessing Hierarchical Input/Output Values

## GetOutputParams, GetOutputParam, GetOutputParamCount

### Syntax: C++

```
VARINFOLIST& GetOutputParams()
```

### Syntax: Visual Basic

```
GetOutputParams() As IPSEnumVarInfo
```

### Syntax: Java

```
PSJVarInfo GetOutputParam(int index)
```

```
int GetOutputParamCount()
```

## Class

C++: InterfaceObject

Visual Basic: IpsIntObj

Java: PSJInterfaceObject

## 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.

## Parameters

None for **GetOutputParams** or **GetOutputParamCount**.

For **GetOutputParam**:

### Parameters

index	The location of the output parameter in the parameter list. Its value can be 0 to (number of parameters - 1).
-------	---

## Return Value

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.



For more information about parameter lists, see [Parameter Lists](#).

---

## 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>

</transaction>
```

```

</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();
```

### Class

C++: CriteriaGroup

### Description

The method **GetRows** gets a CriteriaRowList structure. This structure will contain the criteria rows for a query or update.



For more information about criteria rows and the CriteriaRowList structure, see Understanding Business Interlink Object Criteria.

### Parameters

None.

### Return Value

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

See `GetInterfaceName.GetInterfaceName, InterfaceName`.

## Plug-in Class Methods: Writing a Runtime Plug-in

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.

## ExecuteTransaction

### Syntax: C++

```

virtual EIOCEXECSTATUS ExecuteTransaction(InterfaceObject * IntObj, const int
BatchMode);

```

### Syntax: Visual Basic

```

ExecuteTransaction(IntObj As PsIntObj, BatchMode As Long) As ENUM_EIOCEXECSTATUS

```

### Syntax: Java

```

int ExecuteTransaction(PSJInterfaceObject intobj)

```

## Class

C++: `DLLBaseDriver`

Visual Basic: PsIoDriver

Java: PSJInterfaceObject

## 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.



For more information on writing the ExecuteTransaction method, see Writing the Execution Method for a Runtime Plug-In.

## Parameters

IntObj	An Interlink object. This is provided by the Interlink Framework.
BatchMode	Not currently used.

## Return Value

<b>Value</b>	<b>Meaning</b>
INTOBJ_SUCCEED	Return this if the method succeeds.
INTOBJ_FAILED	Return this if the method fails.

## Example



For more information on an example of the ExecuteTransaction method, see Writing the Execution Method for a Runtime Plug-In and Examples of Business Interlink Runtime Plug-in Code.

## ExecuteObjectQuery

### Syntax: C++

```
virtual EIOCEXECSTATUS ExecuteObjectQuery(InterfaceObject * IntObj, const int BatchMode);
```

### Syntax: Visual Basic

```
ExecuteObjectQuery(ByVal pIntObj As PSIODRIVERLib.IntObj, ByVal BatchMode As Long) As ENUM_EIOCEXECSTATUS
```

**Class**

C++: DLLBaseDriver

Visual Basic: PsIoDriver

**Description**

The ExecuteObjectQuery method is the method that executes the query runtime plug-in. Since ExecuteObjectQuery is a virtual method, you must write the code for this method when you are executing an Interlink Object that is of type query.

**Parameters**

IntObj	An Interlink object. This is provided by the Interlink Framework.
BatchMode	Not currently used.

**Return Value**

<b>Value</b>	<b>Meaning</b>
INTOBJ_SUCCEED	Return this if the method succeeds.
INTOBJ_FAILED	Return this if the method fails.

**Example**

For more information on an example of the ExecuteObjectQuery method, see Writing The Execution Method for a Runtime Plug-In (Criteria Data).

**ExecuteObjectAdd****Syntax: C++**

```
virtual EIOCEXECSTATUS ExecuteObjectAdd(InterfaceObject * IntObj, const int
BatchMode);
```

**Syntax: Visual Basic**

```
ExecuteObjectAdd(IntObj As PsIntObj, BatchMode As Long) As ENUM_EIOCEXECSTATUS
```

**Class**

C++: DLLBaseDriver

Visual Basic: PsIoDriver

## Description

The ExecuteObjectAdd method is the method that executes the add runtime plug-in. Since ExecuteObjectAdd is a virtual method, you must write the code for this method when you are executing an Interlink Object that is of type add.

## Parameters

IntObj	An Interlink object. This is provided by the Interlink Framework.
BatchMode	Not currently used.

## Return Value

<b>Value</b>	<b>Meaning</b>
INTOBJ_SUCCEED	Return this if the method succeeds.
INTOBJ_FAILED	Return this if the method fails.

## Example



For more information about coding ExecuteTransaction, which is very similar to coding ExecuteObjectAdd, see Writing the Execution Method for a Runtime Plug-In.

## ExecuteObjectDelete

### Syntax: C++

```
virtual EIOCEXECSTATUS ExecuteObjectDelete(InterfaceObject * IntObj, const int
BatchMode) ;
```

### Syntax: Visual Basic

```
ExecuteObjectDelete(IntObj As PsIntObj, BatchMode As Long) As
ENUM_EIOCEXECSTATUS
```

## Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

## Description

The ExecuteObjectDelete method is the method that executes the delete runtime plug-in. Since ExecuteObjectDelete is a virtual method, you must write the code for this method when you are executing an Interlink Object that is of type delete.



For more information about writing ExecuteObjectDelete, which uses the Criteria methods, see Writing The Execution Method for a Runtime Plug-In (Criteria Data).

## Parameters

IntObj	An Interlink object. This is provided by the Interlink Framework.
BatchMode	Not currently used.

## Return Value

<i>Value</i>	<i>Meaning</i>
INTOBJ_SUCCEED	Return this if the method succeeds.
INTOBJ_FAILED	Return this if the method fails.

## ExecuteObjectUpdate

### Syntax: C++

```
virtual EIOCEXECSTATUS ExecuteObjectUpdate(InterfaceObject * IntObj, const int BatchMode);
```

### Syntax: Visual Basic

```
ExecuteObjectUpdate(IntObj As PsIntObj, BatchMode As Long) As  
ENUM_EIOCEXECSTATUS
```

## Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

## Description

The ExecuteObjectUpdate method is the method that executes the delete runtime plug-in. Since ExecuteObjectUpdate is a virtual method, you must write the code for this method when you are executing an Interlink Object that is of type update.



For more information about writing ExecuteObjectUpdate, which uses the Criteria methods, see Writing The Execution Method for a Runtime Plug-In (Criteria Data).

### Parameters

IntObj	An Interlink object. This is provided by the Interlink Framework.
BatchMode	Not currently used.

### Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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;
```

### Syntax: Visual Basic

```
PsIoDriver_GetVer() As String
```

### Syntax: Java

```
String GetVersion()
```

### Description

#### Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

Java: PSJInterfaceObject

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.



For more information about these methods, see Writing the Version Methods for a Business Interlink Runtime Plug-In.

## Parameters

None.

## Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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;
```

**Syntax: Visual Basic**

```
PsIoDriver_IsVersionCompatible(ByVal version As String) As Long
```

**Syntax: Java**

```
boolean IsVersionCompatible(String version)
```

**Class**

C++: DLLBaseDriver

Visual Basic: PsIoDriver

Java: PSJInterfaceObject

**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.



For more information and another example about **IsVersionCompatible**, see Writing `IsVersionCompatible` for Your Business Interlink Plug-in Class.

**Parameters**

<i>version</i>	the plug-in version number that is passed in from other Interface Drivers in the PeopleTools database.
----------------	--

**Return Value**

<b>Value</b>	<b>Meaning</b>
BOOL, Long	True if <i>version</i> 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;}
```

## Doc Class Methods: Accessing Hierarchical Input/Output Values

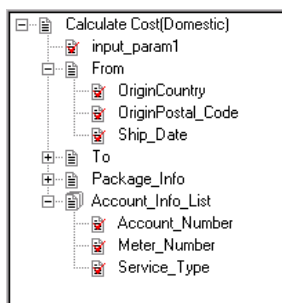
You use the CBIDocs/PsCBIDocs methods to access the inputs and output values of a Business Interlink Object when the inputs and outputs are stored in a document. This is a hierarchical structure, as opposed to flat tables.

For C++, you must include the header file psbidoc.h for this class.

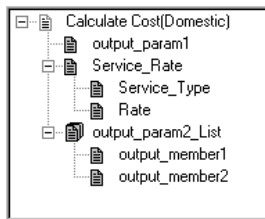
A Business Interlink can have hierarchical data, meaning that the inputs and outputs for a Business Interlink object are stored in the form of a CBIDocs document. Here is an example CBIDocs document: an edited version of the Calculate Cost(Domestic) transaction from the Freight Carrier plug-in.



**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.



Example CBIDocs Input 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.

## AddDoc

### Syntax: C++

```
CBIDocs AddDoc(const TCHAR*)
```

### Syntax: Visual Basic

```
AddDoc(String) As PSBIDocs
```

### Syntax: Java

```
PSJBIDocs AddDoc(String)
```

## Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSJBIDocs

## 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**.

## Parameters

TCHAR*, string	The name of the document that <b>AddDoc</b> adds to the output document.
----------------	--

## Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
CBIDocs, PsBIDocs, PSJBIDocs	The document that <b>AddDoc</b> 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

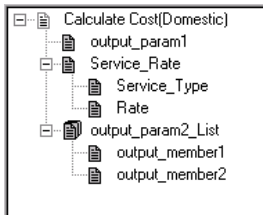
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc

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();
```

### Syntax: Visual Basic

```
AddNextDoc() As Long
```

### Syntax: Java

```
int AddNextDoc();
```

### Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSJBIDocs

### 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.

## Parameters

None.

## Return Value

An integer with the following possible values:

<b>Number</b>	<b>Enum value and Meaning</b>
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

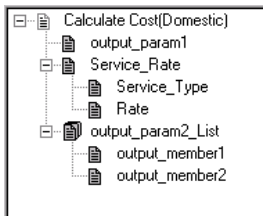
docRate.destroy();

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);

```

### Syntax: 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

```

**Syntax: Java**

```
AddValue(String name, String value);

int AddValue(String name, Integer value);

int AddValue(String name, Float value);

int AddValue(String name, Double value);
```

**Class**

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSBIDocs

**Description**

Adds a value to a member of a document within the output document for a Business Interlink Object.

**Parameters**

<i>name</i>	The name of the member of the document that is having a value added to it.
<i>value</i>	The value that is added.

**Return Value**

An integer with the following possible values:

<b>Number</b>	<b>Enum value and Meaning</b>
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. **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
```

```

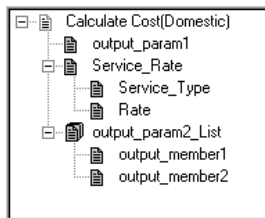
docRate.destroy();

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()
```

### Syntax: Visual Basic

```
Clear()
```

### Syntax: Java

```
Clear()
```

## Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PJSBIDocs

## 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.

**Parameters**

None.

**Return Value**

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****Syntax: Java**

```
PSJBIDocs destroy()
```

**Class**

Java: PSJBIDocs

**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.

**Parameters**

None.

## Return Value

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

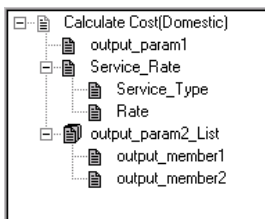
docRate.destroy();

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();
```

### Syntax: Visual Basic

```
Empty() As Long
```

## Class

C++: CBIDocs

Visual Basic: PsBIDocs

## 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.

## Parameters

None.

## Return Value

<b>Value</b>	<b>Meaning</b>
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);
```

### Syntax: Visual Basic

```
GetCount(name As String) As Long
```

## Syntax: Java

```
int GetCount(String name);
```

## Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSJBIDocs

## Description

Returns the number of documents within a document list or parameter list contained within a CBIDocs input document for a Business Interlink Object.

## Parameters

<i>name</i>	The name of the document list or parameter list.
-------------	--

## Return Value

<b>Value</b>	<b>Meaning</b>
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

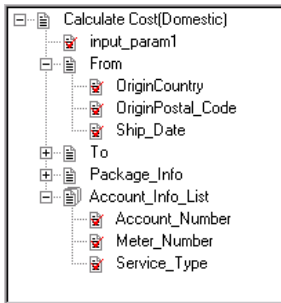
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc

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);
```

### Syntax: Visual Basic

```
GetDoc(docname As String) As PsBIDocs
```

### Syntax: Java

```
PSJBIDocs GetDoc(String docname);
```

## Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSJBIDocs

## 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.

## Parameters

*docname* The name of the input document that **GetDoc** gets.

## Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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) */

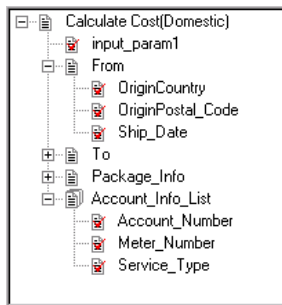
    /* Call the external system and put the resulting values into the CBIDocs output
    documents (code not shown) */

    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(""));

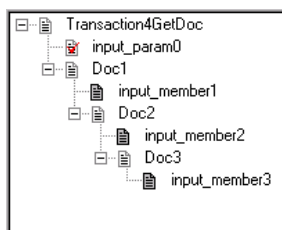
docsIn.ResetCursor();

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()
```

### Syntax: Visual Basic

```
GetNextDoc() As Long
```

### Syntax: Java

```
int GetNextDoc()
```

### Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSJBIDocs

### 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.

### Parameters

None.

### Return Value

An integer with the following possible values:

<b>Number</b>	<b>Enum value and Meaning</b>
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 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

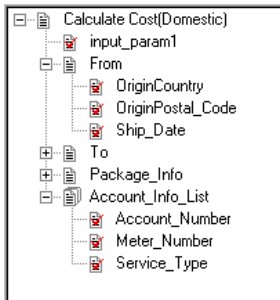
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc

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();
```

### Syntax: Visual Basic

```
GetPreviousDoc() As Long
```

### Syntax: Java

```
int GetPreviousDoc();
```

## Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSJBIDocs

## 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.

## Parameters

None.

## Return Value

An integer with the following possible values:

<b>Number</b>	<b>Enum value and Meaning</b>
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 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

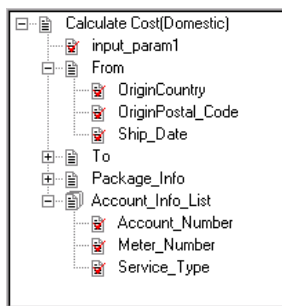
// Use one destroy for each GetInputDocs, GetOutputDocs, AddDoc, GetDoc

```

```
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()
```

### Syntax: Visual Basic

```
GetStatus() As Long
```

### Class

C++: CBIDocs

Visual Basic: PsBIDocs

### Description

Tests the document for a Business Interlink Object. Returns the status of the document.

### Parameters

None.

### Return Value

An integer with the following possible values:

<b>Number</b>	<b>Enum value and Meaning</b>
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

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&);

```

### Syntax: 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

```

### Syntax: 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);
```

## Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSJBIDocs

## Description

Gets a value from an input parameter within a document of the input document for a Business Interlink Object.

## Parameters

*name* The name of the member of the document that is having its value retrieved.

*value*, *int*&, *float*&, *double*&, *time* The retrieved value.

## Return Value

<b>Value</b>	<b>Meaning</b>
<i>value</i> , <i>int</i> &, <i>float</i> &, <i>double</i> &	The value of the input parameter.
<i>int</i>	When the syntax is <code>int GetValue(<i>name</i>, <i>value</i>)</code> , <i>int</i> is the return status of <code>GetValue</code> , listed in the table below.

## Return Value for int

An integer with the following possible values:

<b>Number</b>	<b>Enum value and Meaning</b>
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 <code>GetDoc/GetOutputDocs</code> once, you can call <code>GetNextDoc</code> 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);
```

### Syntax: Visual Basic

```
MoveToDoc(iIndex As Long) As Long
```

### Syntax: Java

```
int MoveToDoc(int iIndex);
```

## Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSJBIDocs

## 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.

## Parameters

*iIndex* is 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.

## Return Value

An integer with the following possible values:

<b>Number</b>	<b>Enum value and Meaning</b>
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 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");

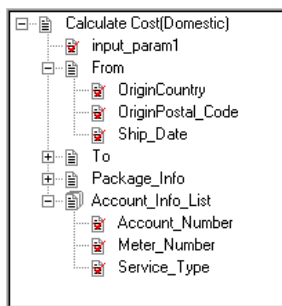
String str3 = docAccountInfo.GetValue("Service_Type");

// 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();
```

### Syntax: Visual Basic

```
ResetCursor()
```

### Class

C++: CBIDocs

Visual Basic: PsBIDocs

Java: PSJBIDocs

### 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.

### Parameters

None.

## Return Value

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();
```

## PSIOString methods: Accessing Input/Output Parameter Information (C++ only)

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 table shows which PSIOString methods you use to operate on the input and output values:

<b>Action to perform:</b>	<b>Use this method</b>
Append a string	Append(PSIOString), operator += or +
To get a TCHAR string from a string	c_str

To find a character in a string	find
To find the length of a string in characters	length
To find a character in a string, starting from the right end of the string	rfind
	size
To get a substring from a string	substr
To perform comparisons on strings	Operators ==, !=

## append

### Syntax: C++

Class: PSIOString

```
CPSIOString& append(PSIOString);
```

```
CPSIOString& append(TCHAR);
```

### Description

Appends a string to the end of the PSIOString string.

### Parameters

TCHAR                                      The string that you want to append to this string.

### Return Value

<b>Value</b>	<b>Meaning</b>
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++

Class: PSIOString

```
char& c_astr();
```

### Description

Converts a PSIOString string to an ASCII string.

The data for a Business Interlink Object is stored entirely in UniCode. If your external system uses ASCII data rather than UniCode data, its functions will require ASCII data. Use the `c_astr` method (instead of the `c_str` method) to pass the Business Interlink data to your external system functions.

### Parameters

None.

### Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
char	The char string, converted from the PSIOString.

### Example

The following code shows the `c_astr` method being used to extract input parameters values in ASCII. This example is an edited version of the UPS 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++

Class: PSIOString

```
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 UniCode data, use the c\_astr method (instead of the c\_str method) to pass the Business Interlink data to your software system functions.

### Parameters

None.

### Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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 VARINFOLIST 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++

Class: PSIOString

```
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.

### Parameters

PSIOString or TCHAR                      The string to search for.

### Return Value

<b>Value</b>	<b>Meaning</b>
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++

Class: PSIOString

```
int length();
```

### Description

Finds the length of this PSIOString in characters.

### Parameters

None.

### Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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++

Class: PSIOString

```
int rfind(PSIOString);

int rfind(TCHAR);
```

### Description

Finds the location of a string within this PSIOString, searching from the right end of the PSIOString.

### Parameters

PSIOString or TCHAR                      The string to search for.

### Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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++

Class: PSIOString

```
int size();
```

### Description

For PSIOString, which contains a UniCode string, returns the number of characters in the string.

For VARINFOLIST, 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.

### Parameters

None.

## Return Value

<b>Value</b>	<b>Meaning</b>
int	<p>For PSIOString, which contains a UniCode string, returns the number of characters in the string.</p> <p>For VARINFOLIST, which contains the input/output parameter information, returns the number of input/output parameters.</p> <p>For IOSTRINGLIST, which contains a row of input/output parameter values, returns the number of input/output parameters.</p>

## 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++

Class: PSIOString

```
PSIOString substr(int);
```

### Description

Returns a string containing the contents of this string, from the location given to the end of the string.



```

        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 +, +=, ==, !=

Class: PSIOString

These operators allow the following operations to be performed upon PSIOString and TCHAR:

++	Append one PSIOString to another PSIOString.
+	Append two PSIOStrings and store the result in a third PSIOString.
==	equal comparison
!=	not equal comparison

## 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:

<b>C++:</b> <b>VARINFOLIST</b> <b>T</b>	<b>Visual Basic:</b> <b>IPsEnumVarInfo</b>	<b>Java: VarInfo</b>	<b>Meaning</b>
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_EIOCDAT ATYPE DataType	int DataType	the data type of the parameter.
EVARATTRI BUTE m_nAttribute	ENUM_VARATT RIBUTE Attribute	int nAttribute()	indicates if the parameter is required.
PSIOString m_strPathNa me	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_strDisplay Name	String DisplayName	String strDisplayNa me	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:

<b>Data Type: C++, Visual Basic, Java</b>	<b>Definition and format</b>
IOC_DATATYPE_INT, ENUM_IOC_DATATYPE_IN T	An integer.
IOC_DATATYPE_STRING, ENUM_IOC_DATATYPE_S TRING	A character string.
IOC_DATATYPE_FLOAT, ENUM_IOC_DATATYPE_F LOAT	A floating point variable.
IOC_DATATYPE_BOOLEA N, ENUM_IOC_DATATYPE B	A Boolean value of TRUE or FALSE.

OOLEAN	
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, ENUM_IOC_DATATYPE_TIME	A time. The format is HH:MM:SS, where HH is the hour, MM is the minutes, and SS is the seconds.
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 GetClassName 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.

# Writing the 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 run-time plug-in. The plug-in is used during the design-time to define and save Business Interlink Definitions. It can also be used at run-time to simulate the execution of the corresponding Business Interlink Objects, if you supply default output values.



---

For more information about writing the run-time plug-in, see [Writing the Execution Method for a Runtime Plug-In](#).

---

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.



---

For more information about writing a design-time plug-in, see [Writing an XML Design-Time Plug-In](#).

---

- 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.
- List any relational or logical operators (used with classes for query and update Business Interlinks): `GetCriteriaLogicalOperators` for logical operators, `GetCriteriaRelationalOperators` for relational operators.

## Writing The Design-Time Methods

This section of the chapter explains the design-time methods that you write for a Visual Basic example, and provides an example for those methods.

### Write 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 str");
}
```



For more information about the `GetDesc` method, see `GetDesc`, `PsIoDriver_GetDesc`.

---

## List 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;  
    }  
}
```



---

For more information about `IsSupported`, see `IsSupported`, `PsIoDriver_IsSupported`.

---

### **For Query and Update Plug-ins, List Relational and Logical Operators: `GetCriteriaRelationalOperators`, `GetCriteriaLogicalOperators`**

If your Business Interlink is of type query or update, write the method `GetCriteriaLogicalOperators` to return a list of the Logical Operators.

If your Business Interlink is of type query or update, write the method `GetCriteriaRelationalOperators` to return a list of the Relational Operators.

Since this Business Interlink Plug-in does not use Relational or Logical Operators, the Visual Basic and C++ examples in this chapter return `False` for these methods.



---

For more information on writing `GetCriteriaRelationalOperators`, `GetCriteriaLogicalOperators`, see `GetCriteriaRelationalOperators`, `PsIoDriver_GetCriteriaRelationalOperators` and `GetCriteriaLogicalOperators`, `PsIoDriver_GetCriteriaLogicalOperators`.

---

### **List 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.



---

For more information about `GetParameterList`, see `GetParameterList`, `PsIoDriver_GetParameterList`.

---

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://ups.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;

}
```

## Write 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 `plstCriteriaNames`. Use the `plstCriteriaNames` method `Add` to add a category name to the list.



For more information about GetCategories, see [GetCategories](#), [PsIoDriver\\_GetCategories](#).

The following example creates the two categories All Transactions and Transactions.

```
Private Function PsIoDriver_GetCategories(ByVal eCatType As
PSIODRIVERLib.ENUM_EIIOCINTERFACESUPPORTED, 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
```

## Write 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.



For a C++ example, see GetClassByName, PsIoDriver\_GetClassByName.

---

```

BOOL SimpleDriver::GetClassByName(const PSIOString& strClassName, ClassDef&
classes)

{

    return FALSE;

}

```

## Write 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`.



For more information about the data structure of the parameter list, see Parameter Lists. For more information about the Add method, see Add.

---

```
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`.



For more information about the data structure of the parameter list, see [Parameter Lists](#). For more information about the `AddInput` and `AddOutput` methods, see [AddInput](#) and [AddOutput](#).

---

```

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;

}

```

## 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

```

## 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 = "UPS Express Box"

objVarInfo.DataType = ENUM_IOC_DATATYPE_ENUM

objVarInfo.DataDescr = "Your Packaging,UPS Letter Envelop,UPS Tube,UPS  
Express Box,UPS Worldwide 25KG Box,UPS 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 = "UPS 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://ups.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
```

## 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 str");
}
```

```
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&
criterialNames)

{

    if(type == IOC_TRANSACTION)

    {

        criterialNames.push_back(_T("Transactions"));

        criterialNames.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: Writing Design-Time Functionality

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 *);
```

### Syntax: Visual Basic

```
FetchNextChunk (IPsEnumString) As Long
```

### Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

### Description

Write this method to return the next chunk of the input or output table.

### Parameters

IpsEnumString	The table from which a chunk is being returned.
InterfaceObject *	The interlink object.

### Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
BOOL	True if a chunk was returned, false otherwise.
Long	

## GetCategories, PsIoDriver\_GetCategories

### Syntax: C++

```
virtual BOOL GetCategories (EIOCINTERFACESUPPORTED type, IOSTRINGLIST& list) = 0;
```

### Syntax: Visual Basic

```
GetCategories (type As ENUM_EIOCINTERFACESUPPORTED, list As IPsEnumString) As Long
```

### Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

## 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.

## Parameters

<i>Type</i>	The interface type for which to list categories.
<i>List</i>	The list of categories that is returned.

## Return Value

<b>Value</b>	<b>Meaning</b>
BOOL	True if a list of categories is returned, False otherwise.
Long	
<i>List</i>	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&
criterialNames)
{
    if(type == IOC_TRANSACTION)
    {
        criterialNames.push_back(_T("Transactions"));
        criterialNames.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;

```

### Syntax: Visual Basic

```

PsIoDriver_GetClassByName(ByVal strClassName As String, ByVal classes As
PSIODRIVERLib.IPsEnumVarInfo) As Long

```

### Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

### 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.

### Parameters

<i>strClassName</i>	A character string. GetClassByName finds all the class names containing this character string.
<i>classes</i>	An array of strings returned that contains the data members for the class named <i>strClassName</i> .

## Return Value

<b>Value</b>	<b>Meaning</b>
<i>classes</i>	An array of strings that contains the data members for the class named <i>strClassName</i> .
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.



For more information about the SetClassName and AddDataMemberDef methods, see SetClassName and AddDataMemberDef.

```

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.



For more information about the Add method, see Add.

---

```

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;

```

### Syntax: Visual Basic

```

PsIoDriver_GetCriteriaRelationalOperators(ByVal strType As String, list As
IPsEnumString) As Long

```

### Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

### 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.

### Parameters

<i>StrType</i>	A character string.
<i>List</i>	An array of strings returned that contains the relational operators for the data member type <i>strType</i> .

### Return Value

<b>Value</b>	<b>Meaning</b>
<i>List</i>	An array of strings that contains the relational operators for the data member type <i>strType</i> .

BOOL	True if a relational operator list was returned, False otherwise.
Long	

### 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;
```

### Syntax: Visual Basic

```
PsIoDriver_GetCriteriaLogicalOperators(ByVal plstOperators As  
PSIODRIVERLib.IPsEnumString) As Long
```

### Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

### 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.

### Parameters

<i>list</i>	An array of strings returned that contains the logical operators for queries.
-------------	---

### Return Value

<b>Value</b>	<b>Meaning</b>
<i>list</i>	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;
```

### Syntax: Visual Basic

```
PsIoDriver_GetDesc() As String
```

### Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

### 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.

### Parameters

None.

### Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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 str");
}
```

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 str"
    '
End Function
```

## GetLastErrorMessage

### Syntax: C++

```
virtual const IODrvMsg& GetLastErrorMessage () { return m_ErrMsg; }
```

### Syntax: Visual Basic

```
GetLastErrorMessage(pDrvMsg As IPsDriverMessage)
```

**Class**

C++: DLLBaseDriver

Visual Basic: PsIoDriver

**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.

**Parameters**

None.

**Return Value**

<i><b>Value</b></i>	<i><b>Meaning</b></i>
m_ErrMsg, pDrvMsg	The returned error message.

**GetClassByCategory, PsIoDriver\_GetClassByCategory****Syntax: C++**

```
virtual BOOL GetClassByCategory(const PSIOString& strFilter, const PSIOString&
    strCategory, IOSTRINGLIST& list) = 0;
```

**Syntax: Visual Basic**

```
PsIoDriver_GetClassByCategory(strFilter As String, strCategory As String, list
    As IPsEnumString) As Long
```

**Class**

C++: DLLBaseDriver

Visual Basic: PsIoDriver

**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.

## Parameters

<i>strFilter</i>	A string that you can search on, such as "sales" to find all transactions/classes with "sales" in their names.
<i>strCategory</i>	A string containing the category.
<i>list</i>	The array of strings returned that contains the class names (or descriptions) in this category.

## Return Value

<b>Value</b>	<b>Meaning</b>
<i>list</i>	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;
```

### Syntax: Visual Basic

```
PsIoDriver_GetParameterList(list As IPsEnumVarInfo) As Long
```



## Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

## 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.

## Parameters

<i>list</i>	An array of strings containing the configuration parameters.
-------------	--

## Return Value

<b>Value</b>	<b>Meaning</b>
<i>list</i>	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.



For more information about the parameter data structure, see Parameter Lists.

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.



For more information about the `Add` method, see `Add`.

---

```

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;

```

## Syntax: Visual Basic

```
PsIoDriver_GetTransactionByCategory(strFilter As String, bstrCriteria As String,
TransNames As IPsEnumString) As Long
```

## Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

## 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.

## Parameters

<i>strFilter</i>	A string that you can search on, such as "sales" to find all transactions/classes with "sales" in their names.
<i>strCriteria</i>	A string containing the category.
<i>transNames</i>	The array of strings returned that contains the transaction names (or descriptions) in this category.

## Return Value

<b>Value</b>	<b>Meaning</b>
<i>transNames</i>	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.



For more information about the push\_back method, see

---

```
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;

```

### Syntax: Visual Basic

```

PsIoDriver_GetTransactionByName(strTransName As String, plstInput As
IPsEnumVarInfo, plstOutput As IPsEnumVarInfo) As Long

```

### Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

### 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.

### Parameters

*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*

## Return Value

<b>Value</b>	<b>Meaning</b>
<i>TransDef</i>	The array of strings that contains the input/output parameters for the transaction named <i>strTransName</i> .
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.



For more information about the AddInput and AddOutput methods, see AddInput and AddOutput.

```

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.



For more information about the Add method, see Add.

---

## IsSupported, PsIoDriver\_IsSupported

### Syntax: C++

```
virtual BOOL IsSupported(EIOCINTERFACETYPE type) = 0;
```

## Syntax: Visual Basic

```
PsIoDriver_IsSupported(ByVal Type As PSIODRIVERLib.ENUM_EIOCINTERFACESUPPORTED)
As Long
```

## Class

C++: DLLBaseDriver

Visual Basic: PsIoDriver

## 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.

## Parameters

*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_STATICCATEGORY(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)
```

## Return Value

<b>Value</b>	<b>Meaning</b>
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
    End Select
End Function

```

```

        Case Else

            PsIoDriver_IsSupported = False

        End Select

    ,

End Function

```

## Transaction/Class Methods: Adding Input/Output Parameters to Transactions, Data Members to Classes

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 table shows which methods you use to operate on the input and output tables:

<b>Action to perform:</b>	<b>Use this method</b>
Add a parameter to a transaction or a data member to a class in Visual Basic	Add
Add a data member to a class in C++	AddDataMemberDef
Add an input parameter to a transaction in C++	AddInput
Add an output parameter to a transaction in C++	AddOutput
Set a class name in C++	SetClassName
Set a transaction name in C++	SetTransactionName

### Add

#### Syntax: Visual Basic

```
Add ()
```

#### Class

IPsEnumVarInfo

#### 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 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.



For more information about the parameter data structure and the data types, see Parameter Lists.

---

- 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.

## Return Value

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++

Class: TransactionDef

```
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 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.



For more information about the parameter data structure and the data types, see Parameter Lists.

### Parameters

*var* the variable that is added as an input parameter.

### Return Value

<b>Value</b>	<b>Meaning</b>
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++

Class: ClassDef

```

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 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.



For more information about the parameter data structure and the data types, see Parameter Lists.

## Parameters

*var* the variable that is added as a data member.

## Return Value

<b>Value</b>	<b>Meaning</b>
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++

Class: TransactionDef

```
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 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.



For more information about the parameter data structure and the data types, see Parameter Lists.

---

### Parameters

*var* the variable that is added as an input parameter.

## Return Value

<i><b>Value</b></i>	<i><b>Meaning</b></i>
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++

Class: ClassDef

```
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 C++ **SetClassName** method set the name of a class.

### Parameters

*StrClassName*                      the name of the class.

### Return Value

<b>Value</b>	<b>Meaning</b>
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++

Class: TransactionDef

```

    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 C++ **SetTransactionName** method set the name of a transaction.

### Parameters

*StrTransactionName*                      the name of the transaction.

### Return Value

<b>Value</b>	<b>Meaning</b>
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;

}

```

## 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.

## Return Value

None.

## Example

The following C++ code pushes a transaction category.

```
BOOL SimpleDriver::GetCategories(EIOCINTERFACESUPPORTED type, IOSTRINGLIST&
criterialNames)
{
    if(type == IOC_TRANSACTION)
    {
        criterialNames.push_back(_T("Transactions"));
        criterialNames.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;
}
```

# Writing The Execution Method for a Runtime Plug-In (Criteria Data)

Writing the execution method that uses criteria consists of writing one or more of the following methods:

- **ExecuteObjectQuery** for a Business Interlink Query. This method extracts criteria from the Interlink Object, and adds outputs to it.
- **ExecuteObjectUpdate** for a Business Interlink Update. This method extracts inputs and criteria from the Interlink Object.
- **ExecuteObjectDelete** for a Business Interlink Delete. This method extracts criteria from the Business Interlink Object.



For more information about the methods that are mentioned here, refer to *Understanding The Business Interlink Methods*.

---

When you write the execution method, you will use methods to perform the following tasks:

The execution method that you write performs the following tasks. You may or may not do these steps in the following order, depending upon how you wish to write your execution method.

1. Takes as input a Business Interlink Object.
2. Use the **GetObjName** method to extract the name of the Business Interlink Object. The name of the Business Interlink Object is the name of the transaction, and you use it to determine which transaction you are writing.
3. Use the **GetConfigParameters** method to extract the configuration parameter values for this Business Interlink Object, and performs whatever actions are needed with them (usually configuration parameters are needed to connect to the external system).
4. Use the **GetOutputTable** method to get the output table for the Business Interlink Object. Use the **Clear** method to clear the output table.
5. Use the **GetOutputParams** method to get the output parameters, and then use the **AddColumn** method to add one column to the table for each output parameter.

6. If you are writing `ExecuteQuery` or `ExecuteUpdate`, rather than extract input values from the input table, you extract the criteria from the criteria structure instead of performing steps 7 and 8.
7. If you are writing `ExecuteTransaction`, `ExecuteObjectAdd`, or `ExecuteObjectDelete`, use the `GetInputTable` method to get the input table for the Business Interlink Object, and then use the `ResetCursor` method to reset the cursor to the top of the input table.
8. Use the `GetInputParams` method to get the input parameters, and then use the `GetNextRow` method to extract one row of input parameters.
9. Pass your inputs (or the criteria) to the external system, calling functions in the external system to perform the actions that you want your Business Interlink Object to take, and receive the outputs from your external system.
10. Use the `push_back` method to insert the output values into the Business Interlink Object output table.

## Understanding Business Interlink Object Criteria

Criteria are used with the object query Business Interlink Object and the object update Business Interlink Object; the method you write is `ExecuteObjectQuery` or `ExecuteObjectUpdate`. For example, a query may resemble the following:

```
Select output_list from class
    where criteria_rows
```

<i>output_list</i>	a list of the outputs for this query.
<i>class</i>	the class upon which this query is being performed.
<i>criteria_rows</i>	rows of criteria for this query.

A common way for you to build a query within the `ExecuteObjectQuery` method is to create a long character string that forms one query statement. To build that statement, you will need to get the following information from the Business Interlink Object that is the input for your `ExecuteObjectQuery` method:

- The criteria rows. Each row consists of a left-hand-side input, a relational operator, a right-hand-side input, and a logical operator. The rows can also be grouped, as in the following example. Get the rows and the grouping with the `InterfaceObject` criteria structures and methods.

```
Select output from class where A > 5 AND (B < 4 OR C != 10)
```

The rows `B < 4 OR` and `C != 10` have been grouped.

- Outputs. Get the outputs, or the members of the class chosen for output, with the



InterfaceObject method GetOutputParams.

- Class. Get the name of the class upon which the query is being performed with the InterfaceObject method GetObjName.

To understand the criteria rows, and how their data is stored, you should understand the CriteriaRow structure and the CriteriaNode structure.

## Understanding the CriteriaRow Structure for Criteria Rows

A criteria row consists of the following:

```
left-hand-side relational-operator right-hand-side logical-operator
```

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

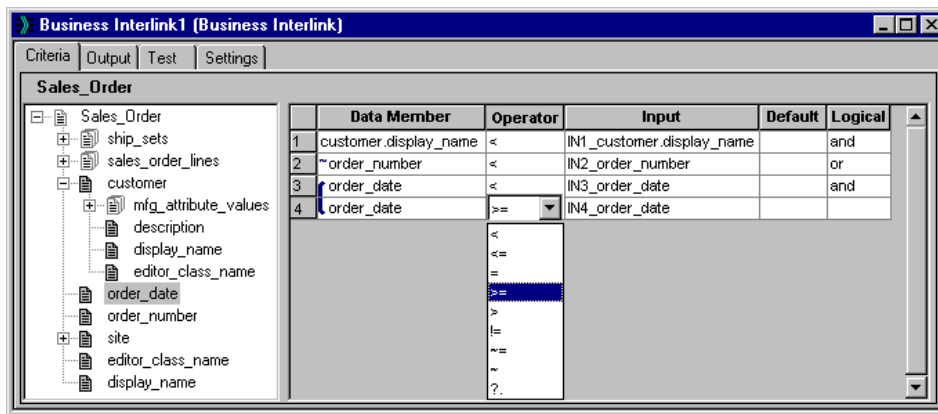
PSIOString dataType;   // the data type
```

Each CriteriaRow structure in the CriteriaRowList is filled within the Application Designer, when a user designs a query Business Interlink Definition and creates the associated PeopleCode program. For example, the following query Interlink Definition designed in the Application Designer on a Sales Order class would produce a query Business Interlink Object that has a CriteriaRowList structure with four CriteriaRows.

The left-hand-side consists of members of the class upon which the query is being performed. An Application Developer, when creating a Business Interlink Definition within the Application Designer, selects these members from the class. The right-hand-side (IN1\_customer.display\_name, IN2\_order\_number, IN3\_order\_date, and IN4\_order\_date) is replaced within PeopleCode with constants, or with names of PeopleCode variables or Record Fields or other legitimate names. The Application Developer, when writing a PeopleCode program for this Business Interlink Definition, selects the constants, PeopleCode variables, etc.

```
Select Sales_Order
where
    customer.display_name = IN1_customer.display_name AND
    NOT order_number < IN2_order_number OR
```

```
(order_date < IN3_order_date AND
order_date >= IN4_order_date)
```



Business Interlink Page: Criteria Tab

## Understanding the CriteriaNode Structure for Negation and Grouping

The CriteriaNode structure is also included with the query Business Interlink Object that is input for the ExecuteObjectQuery method.

The CriteriaNode structure consists of pointers to CriteriaRows in a CriteriaRowList, and also can contain pointers to another CriteriaNode. The CriteriaNode structure has the following form:

```
int type;

BOOL negation;

Union UNION

{

    CriteriaGroup *pGroup;

    Int row;

} data;
```

\*pGroup is a pointer to a CriteriaNodeList. row is a number pointing to a CriteriaRow in the CriteriaRowList structure. To tell you which of these is being used for this Criteria Node, the class CriteriaGroup contains an enum of the form:

```
enum { ELEMENT, GROUP};
```

ELEMENT says that this node uses the pointer to a CriteriaRow, and GROUP says that this node uses the pointer to a CriteriaNodeList.

## Example of CriteriaRowList and CriteriaNodeList

There is a CriteriaRowList structure that contains all of the CriteriaRows. In addition, there is a CriteriaNode list structure that contains at least one CriteriaNodeList containing CriteriaNodes that point to CriteriaRows, plus an additional CriteriaNodeList to point to the CriteriaRows (and CriteriaNodes) contained in each group.

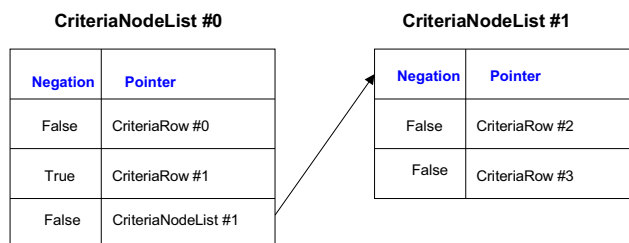
A Business Interlink Object for the following query has four CriteriaRows in its CriteriaRowList. The object also has two CriteriaNodeLists, because there is one grouping within this query.

```
Select Sales_Order
where
  customer.display_name = "Harvey" AND
  NOT order_number < "101" AND
  (order_date < "1999-10-20" OR
   order_date >= "1999-08-20")
```

The query Business Interlink Object contains a CriteriaRowList with four CriteriaRows.

<b>CriteriaRow #</b>	<b>Left hand side lhsExpr</b>	<b>relational operator rOper</b>	<b>Right hand side rhsExpr</b>	<b>Logical operator lOper</b>
0	customer.display_name	=	"Harvey"	AND
1	order_number	<	"101"	AND
2	order_date	<	"1999-10-20"	OR
3	order_date	>=	"1999-08-20"	

The query Business Interlink Object contains two CriteriaNodeLists, each containing CriteriaNodes that point to CriteriaRows or to CriteriaNodeLists. The following diagram shows the structure of the CriteriaNode Lists for this Business Interlink Object. Each node in the CriteriaNodeLists have a negation: the node pointing to CriteriaRow #1 has a negation. CriteriaNodeList #0 contains pointers to CriteriaRow #0 and CriteriaRow #1, and to CriteriaNodeList #1. CriteriaNodeList #1 contains the grouped CriteriaRows #2 and #3.



CriteriaNodeLists

## Understanding m\_CriteriaGroup

m\_CriteriaGroup is a class of type CriteriaGroup. It contains all the CriteriaRow and the CriteriaNode List structures for this Business Interlink Object.

## Building A Query String Using The Criteria Methods and Structures

If your execution method is either ExecuteObjectQuery or ExecuteObjectUpdate, you will use the criteria structures. The most common way to create a query or update is to create a long character string, fill the string with the criteria and other text for a query or update, and to send that string to your external system.

This section will use a character buffer of type TCHAR to contain this string:

```
TCHAR buf[512]
```

### Adding the Outputs to the Query String

Use the InterfaceObject method GetOutputParams to return the output names for this query Business Interlink Object. Then retrieve the output names and use them to build the output names in your query.



For more information on GetOutputParams, see GetOutputParams, GetOutputParam, GetOutputParamCount.

---

The following example code places the string “select” into the output table, retrieves the output names, and then puts the output names into the output table, separating each output name with a comma.

```
_tcscpy(buf, _T("select "));

const VARINFOLIST& varListOutput = IntObj->GetOutputParams();

for(int i = 0; i < varListOutput.size() - 2; i++)
{
    _tscat(buf, varListOutput[i]->m_strName.c_str());

    if(i != varListOutput.size()-3) _tscat(buf, _T(", "));
}
```

`_tcscopy` is a function that copies a character string into a character string buffer. `_tcscat` is a function that concatenates two character strings.

`VARINFOLIST` is a list of `varListOutputs`; a `varListOutput` is a character string containing one output name. The `size VARINFOLIST` method gets the number of output names in this list of output names from `GetOutputParams`. The `c_str PSIOBuf` method gets a UNICODE string out of `varListOutput`; the `c_astr PSIOBuf` method gets an ASCII string.



For more information on the `VARINFOLIST` object, see `Parameter Lists`.

---



For more information about the `size` method, see `size`.

---



For more information about the `c_str` and `c_astr` methods, see `c_str` and `c_astr`.

---

## Adding the Class Name to the Query String

Suppose you have already created, within a `TCHAR buf[512]` character buffer, the character string “select *output1*, *output2*”, where *output1*, *output2* is a comma-delimited character string of all the output names for this query. For this query, you want to add to the buffer the string “from *class*”, where *class* is the name of the class upon which this query will be performed. The name that you want to use is the name of this Business Interlink Object; use the `GetObjName InterfaceObject` method to get it. The following code will add “from *class*” to the buffer.

```
_tcscat(buf, _T(" from "));  
  
_tcscat(buf, IntObj->GetObjName().c_str());
```



For more information on using `GetObjName`, refer to `GetObjName`, `ObjName`.

---

## Adding the Criteria Rows and Groupings to the Query String

You will use the `CriteriaGroup` methods `GetRows()` and `GetGroup()` to get the `CriteriaRows`, and then you will use the `CriteriaRow` structure to extract the left-hand-side, relational-operator, right-hand-side, logical-operator) from the `CriteriaRows`.

## Getting The Criteria Rows: GetRows

Call the GetRows method to get a criteria row.

```
CriteriaRowList* rowlist = IntObj->m_CriteriaGroup.GetRows()
```

*IntObj* the InterfaceObject pointer that is input to this ExecuteTransaction method.

*rowlist* a pointer of type CriteriaRowList. Points to a CriteriaRow within the CriteriaRowList structure.

For 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.

```
GenerateCriteria(strCriteria, &IntObj->m_CriteriaGroup,
    IntObj->m_CriteriaGroup.GetRows());
```

## Getting the Criteria GetGroup()

Call the GetGroup method to get a criteria node.

```
CriteriaNodeList* nodelist =
    &IntObj->m_CriteriaGroup.GetGroup()
```

*IntObj* the InterfaceObject pointer that is input to this ExecuteTransaction method.

*nodelist* a pointer of type CriteriaNodeList. Points to the first Criteria Node in the CriteriaNodeList.

For 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.

```
GenerateCriteria(strCriteria, &IntObj->m_CriteriaGroup,
    IntObj->m_CriteriaGroup.GetRows());
```

```

void RecordDriver::GenerateCriteria(PSIOString& strCriteria,
    CriteriaGroup* Group, CriteriaRowList *rowList)
{
    CriteriaNodeList *gList = Group->GetGroup();

```

## Getting the Elements From The Criteria Rows

Each criteria row consists of a left-hand-side input, a relational operator, a right-hand-side input, and a logical operator (except for the last criteria row in the CriteriaRowList structure, which contains no logical operator).

The following example method, GenerateCriteria, returns a PSIOString character string, strCriteria. strCriteria contains all of the criteria for the query, the relational and logical operators, and the groupings.

GenerateCriteria has two inputs: a pointer to a CriteriaNodeList structure, and a pointer to a CriteriaRowList structure.

Comments within the code below explain how this example method parses the CriteriaNodeList structures, the CriteriaRow structures, and builds the character string from the elements in each CriteriaRow structure.

```

void RecordDriver::GenerateCriteria(PSIOString& strCriteria, CriteriaGroup*
    Group, CriteriaRowList *rowList)
{
    // Get the pointer to a CriteriaNodeList.
    CriteriaNodeList *gList = Group->GetGroup();

    PSIOString op;

    // Loop for the number of CriteriaNodes in the CriteriaNodeList
    for(int i = 0; i < gList->size(); i++)
    {
        // If this CriteriaNode contains a pointer to a CriteriaRow,
        // process the CriteriaRow
        if((*gList)[i].type == CriteriaGroup::ELEMENT)
        {
            // Get the pointer to the CriteriaRow

```

```

CriteriaRow* cRow = (*rowList)[(*gList)[i].data.row-1];

// Add the logical operator for the previous CriteriaNode
// to strCriteria.

strCriteria += op;

// If this CriteriaNode has a negation,
// add NOT ( to strCriteria.

if((*gList)[i].negation) strCriteria += _T(" NOT (");

strCriteria += _T(" ");

// Add the left-hand-side to strCriteria

strCriteria += cRow->lhsExpr;

strCriteria += _T(" ");

// Add the relational operator to strCriteria

strCriteria += cRow->rOper;

strCriteria += _T(" ");

// If the dataType of this CriteriaRow is string, add ' to
strCriteria

if(cRow->dataType == _T("string")) strCriteria += _T(" '");

// Add the right-hand-side to strCriteria

strCriteria += cRow->rhsDefault;

// If the dataType of this CriteriaRow is string, add ' to
strCriteria

if(cRow->dataType == _T("string")) strCriteria += _T("'");

// If this CriteriaNode has a negation,
// add ) to strCriteria; with the previous NOT(, this
// encloses the node within NOT().

if((*gList)[i].negation) strCriteria += _T(") ");

strCriteria += _T(" ");

// Store the logical operator in the op buffer.

op = cRow->lOper;

```



```

    }

    // Else this CriteriaNode contains a pointer to a
    // CriteriaNodeList; process the CriteriaNodeList
    else
    {
        // Add the logical operator for the previous CriteriaNode
        // to strCriteria.

        strCriteria += op;

        // If this CriteriaNode has a negation,
        // add NOT to strCriteria.

        if((*gList)[i].negation) strCriteria += _T(" NOT ");

        // Add ( to strCriteria to start this CriteriaNodeList
        strCriteria += _T(" (");

        // Call GenerateCriteria, using (*glist)[i].data.pGroup as
        // the address of the CriteriaGroup, and using the same
        // pointer to the CriteriaRowList structure that was
        // originally passed into GenerateCriteria.

        GenerateCriteria(strCriteria, (*gList)[i].data.pGroup,
            rowList);

        // Add ) to strCriteria to end this CriteriaNodeList
        strCriteria += _T(") ");
    }
}
}

```



# Index

## A

- Add 9-56
- Add Project Configuration
  - C++ 2-2
- AddDataMemberDef 9-61
- AddDoc 8-34
  - using in example 4-11
- AddInput 9-60
- AddNextDoc 8-39
  - using in example 4-11
- AddOutput 9-63
- AddValue 8-45
  - using in example 4-11
- AddValueDouble 8-45
- AddValueFloat 8-45
- AddValueInt 8-45
- append 8-87
- Architecture 1-1

## B

- Business Interlink
  - actions when running 1-2
  - getting description 8-2
- Business Interlink Architecture 1-1
- Business Interlink Definition
  - getting name of 8-4
- Business Interlink object
  - getting name 8-16
- Business Interlink Object
  - actions taken to create 1-3
  - getting name of 4-2
- Business Interlink type
  - getting 8-15

## C

- c\_astr 8-88
  - C++ use with GetConfigParam 8-7
- c\_str 8-89
  - C++ use with GetConfigParam 8-7
- C++
  - Add Project Configuration 2-2
  - creating a project in C++ under UNIX 2-7
  - creating a project under Windows NT 2-1
  - ExecuteTransaction example 5-1
  - instantiating driver instance 3-3

- New Project dialog box 2-1
- Project Settings for C/C++ 2-3
- Project Settings for Links 2-4
- Set Active Configuration 2-3
- template location 2-6
- categories
  - creating for dynamic catalogs 9-33
- CD-ROM
  - ordering iii
- class data members
  - adding one for dynamic catalogs for C++ 9-61
  - creating for dynamic catalogs 9-35
- class name
  - creating for dynamic catalogs for C++ 9-65
- class names
  - listing for dynamic catalogs 9-44
- Clear 8-51
  - using in example 4-4
- ConfigParam 8-5
- configuration parameters
  - creating for dynamic catalogs 9-46
  - data structure 8-96
  - getting 8-7
  - getting one parameter 8-5

## D

- Definition XML tag 7-4
- deploying the runtime plug-in 6-1
- Description 8-2
- description for Business Interlink
  - creating for dynamic catalogs 9-42
- design-time functionality
  - see dynamic catalogs 9-1
- dynamic catalogs
  - adding one class data member for C++ 9-61
  - adding one class data member for Visual Basic 9-56
  - adding one transaction input parameter for C++ 9-60
  - adding one transaction input/output parameter for Visual Basic 9-56
  - adding one transaction output parameter for C++ 9-63
  - creating categories 9-33
  - creating class data members 9-35
  - creating class names for C++ 9-65
  - creating configuration parameters 9-46
  - creating description for Business Interlink 9-42
  - creating logical operators 9-41
  - creating relational operators 9-39

- creating transaction input/output parameters 9-51
- creating transaction names for C++ 9-66
- listing class names for a category 9-44
- listing transaction names for a category 9-48
- writing 9-1

## E

- Empty 8-54
- error message
  - getting last one 9-43
- ExecuteObjectAdd 8-27
  - writing 4-1
- ExecuteObjectDelete 8-28
- ExecuteObjectQuery 8-26
- ExecuteObjectUpdate 8-29
- ExecuteTransaction 8-25
  - C++ example 5-1
  - Java example 5-17
  - Visual Basic example 5-10
  - writing 4-1
- Execution Method
  - writing 4-1

## F

- FetchNextChunk 9-33
- find 8-90, 8-92

## G

- GetCategories 9-33
- GetClassByCategory 9-44
- GetClassByName 9-35
- GetConfigParam 8-5
- GetConfigParamCount 8-5
- GetConfigParams 8-6
- GetCount 8-55
- GetCriteriaRelationalOperators 9-39
- GetDesc 9-42
- GetDescription 8-2
- GetDoc 8-60
  - using in example 4-7
- GetGroup 8-8
- GetInputDocs 8-9
  - using in example 4-3
- GetInputParam 8-11
- GetInputParamCount 8-11
- GetInputParams 8-11
- GetInterfaceName 8-3
- GetInterfaceType 8-15
- GetLastErrorMessage 9-43
- GetLogicalRelationalOperators 9-41
- GetNextDoc 8-65
  - using in example 4-7

- GetObjName 8-16
  - writing 4-2
- GetOutputDocs 8-18
  - using in example 4-4
- GetOutputParam 8-20
- GetOutputParamCount 8-20
- GetOutputParams 8-20
  - using in example 4-4
- GetParameterList 9-46
- GetPreviousDoc 8-70
- GetRows 8-24
- GetStatus 8-75
- GetTransactionByCategory 9-48
- GetTransactionByName 9-51
- GetValue 8-77
  - using in example 4-7
- GetValueDouble 8-77
- GetValueFloat 8-77
- GetValueInt 8-77
- GetVer 8-30
- GetVersion 8-30
  - writing 3-1

## H

- Header XML tag 7-4

## I

- input document
  - adding 8-60
  - getting 4-3, 8-9
  - getting all input documents and the values in a loop 4-7
  - getting number of documents 8-55
  - getting the next document in a list 8-65
  - getting the previous document in a list 8-70
  - getting values 8-77
  - moving to a document in a list 8-82
  - resetting to the first 8-85
- input parameters
  - data structure 8-96
  - getting 8-11
- Inputs XML tag 7-4
- InstantiateDriverInstance
  - writing for C++ 3-3
- interface type
  - setting supported types 9-53
- InterfaceName 8-3
- InterfaceName XML tag 7-4
- InterfaceType 8-15
- IoCollection.DLL
  - registering for Visual Basic 2-11
- IOSTRINGLIST
  - getting size of 8-93
- IPsEnumVarInfo

- data structure 8-96
- IsSupported 9-53
- IsVersionCompatible 8-31
- writing 3-2

## J

- Java
  - creating a project under UNIX 2-19
  - creating a project under Windows NT 2-17
  - ExecuteTransaction example 5-17

## L

- length 8-91
- logical operators
  - creating for dynamic catalogs 9-41

## M

- method
  - Add 9-56
  - AddDataMemberDef 9-61
  - AddDoc 8-34
  - AddInput 9-60
  - AddNextDoc 8-39
  - AddOutput 9-63
  - AddValue 8-45
  - AddValueDouble 8-45
  - AddValueFloat 8-45
  - AddValueInt 8-45
  - append 8-87
  - c\_astr 8-88
  - c\_str 8-89
  - Clear 8-51
  - ConfigParam 8-5
  - Description 8-2
  - Empty 8-54
  - ExecuteObjectAdd 8-27
  - ExecuteObjectDelete 8-28
  - ExecuteObjectQuery 8-26
  - ExecuteObjectUpdate 8-29
  - ExecuteTransaction 8-25
  - FetchNextChunk 9-33
  - find 8-90, 8-92
  - GetCategories 9-33
  - GetClassByCategory 9-44
  - GetClassByName 9-35
  - GetConfigParam 8-5
  - GetConfigParamCount 8-5
  - GetConfigParams 8-6
  - GetCount 8-55
  - GetCriteriaRelationalOperators 9-39
  - GetDesc 9-42
  - GetDescription 8-2

- GetDoc 8-60
- GetGroup 8-8
- GetInputDocs 8-9
- GetInputParam 8-11
- GetInputParamCount 8-11
- GetInputParams 8-11
- GetInterfaceName 8-3
- GetInterfaceType 8-15
- GetLastErrorMessage 9-43
- GetLogicalRelationalOperators 9-41
- GetNextDoc 8-65
- GetObjName 8-16
- GetOutputDocs 8-18
- GetOutputParam 8-20
- GetOutputParamCount 8-20
- GetOutputParams 8-20
- GetParameterList 9-46
- GetPreviousDoc 8-70
- GetRows 8-24
- GetStatus 8-75
- GetTransactionByCategory 9-48
- GetTransactionByName 9-51
- GetValue 8-77
- GetValueDouble 8-77
- GetValueFloat 8-77
- GetValueInt 8-77
- GetVer 8-30
- GetVersion 8-30
- InterfaceName 8-3
- InterfaceType 8-15
- IsSupported 9-53
- IsVersionCompatible 8-31
- length 8-91
- MoveToDoc 8-82
- ObjName 8-17
- ResetCursor 8-85
- SetClassName 9-65
- SetTransactionName 9-66
- size 8-93
- substr 8-94
- MoveToDoc 8-82

## N

- New Project dialog box
  - Visual Basic 2-1

## O

- ObjName 8-17
- output document
  - adding 8-34
  - adding all output documents and the values in a loop 4-11
  - adding the next document to a list 8-39
  - adding values 8-45

- clearing 8-51
- getting 4-4, 8-18
- testing to see if empty 8-54
- output parameters
  - data structure 8-96
  - getting 8-20
  - getting in example 4-4

## P

- PeopleBooks
  - CD-ROM, ordering iii
  - printed, ordering iii
- project
  - creating for C++ under UNIX 2-7
  - creating for C++ under Windows NT 2-1
  - creating for Java under UNIX 2-19
  - creating for Java under Windows NT 2-17
  - creating for Visual Basic 2-12
- Project Settings for C/C++ 2-3
- Project Settings for Links
  - C++ 2-4
- PsIntObj.DLL
  - registering for Visual Basic 2-11
- PsIodriver
  - selecting as class for Visual Basic 2-15
- PSIOString
  - appending characters 8-87
  - finding characters 8-90
  - finding characters from right end 8-92
  - getting length of 8-91
  - getting size of 8-93
  - getting substring from 8-94
  - operators 8-96
  - passing ASCII data to external system functions 8-88
  - passing UniCode data to external system functions 8-89

## R

- References
  - setting for Visual Basic 2-13
- relational operators
  - creating for dynamic catalogs 9-39
- ResetCursor 8-85
  - using in example 4-3
- runtime plug-in
  - deploying 6-1
  - placement for testing 6-1

## S

- Set Active Configuration

- C++ 2-3
- SetClassName 9-65
- SetTransactionName 9-66
- size 8-93
- status
  - getting for CBIDocs/PsBiDocs methods 8-75
- substr 8-94

## T

- template
  - location for C++ 2-6
- testing
  - placing the runtime plug-in for testing 6-1
- transaction
  - getting name of 4-2
- transaction input parameters
  - adding one for dynamic catalogs for C++ 9-60
- transaction input/output parameters
  - adding one for dynamic catalogs for Visual Basic 9-56
  - creating for dynamic catalogs 9-51
- transaction name
  - creating for dynamic catalogs for C++ 9-66
- transaction names
  - listing for dynamic catalogs 9-48
- transaction output parameters
  - adding one for dynamic catalogs for C++ 9-63

## V

- VARINFOLIST
  - data structure 8-96
  - getting size of 8-93
- version
  - creating version number 3-1, 8-30
  - setting compatability 3-2, 8-31
- Visual Basic
  - creating a project 2-12
  - ExecuteTransaction example 5-10
  - registeringIoCollection.DLL and PsIntObj.DLL 2-11
  - selecting PsIoDriver as implement class 2-15
  - setting References 2-13

## X

- XML tag
  - Definition 7-4
  - Header 7-4
  - Inputs 7-4
  - InterfaceName 7-4