WebSphere® Partner Agreement Manager

Adapter Developer's Guide

Version 2 Release 2

BIAAAD02

Note: Before using this information and the product it supports, read the information in *Notices* on page 295.

Third Edition (July 2001)

This edition applies to version 2, release 2 of WebSphere Partner Agreement Manager (product number 5724-A85) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can make comments on this information via e-mail at idrcf@hursley.ibm.com.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2000-2001. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

TABLE OF CONTENTS

Welcome to the Adapter Developer's Guide

This document describes WebSphere® Partner Agreement Manager 2.2 and explains how to use Partner Agreement Manager's adapter technology as a bridge between Partner Agreement Manager processes and specific business applications interfaces.

To use adapters with your applications:

- Read *Introducing adapters* on page 1 for an overview of Partner Agreement Manager adapter technology and the Partner Agreement Manager adapter development environment.
- See *Using an integration wizard* on page 9 for information on configuring integration wizards and a tutorial on how to use an integration wizard to create an adapter for a flat file.
- See *Designing Adapters* on page 23 for information about designing custom adapters.
- See *Creating an adapter type* on page 41 for information about creating adapter types and importing and exporting adapter types and implementations.
- See *Creating Adapter Implementations* on page 61 for information about creating adapter implementations in the Adapter Designer.
- See *Using the JDBC Integration Wizard* on page 69 for information about using the JDBC integration wizard to create an adapter for Oracle or SQL Server.

- See *Using utility adapters* on page 83 for descriptions of the Partner Agreement Manager utility adapters and the business objects they use.
- See *Using adapter API methods* on page 103 for reference information on the Adapter API methods that you can use in adapter code.
- See *Using Business Object API methods* on page 191 for information on using the business object API methods that you can use in your adapter code.
- See *Adding custom code to adapters* on page 247 for information about adding custom code to adapter implementations.
- See *Java implementation example* on page 257 for a description of the sample Inventory adapter that is included in the Adapter Development Environment.

Who should use this information

This information is for those who need to develop adapters. Adapters give private processes a uniform way to access the capabilities of a variety of external applications.

Related information

For additional information see the following:

- The readme.htm file. This file may contain information that became available after this book was published. Before installation, the readme.txt file is located in the root directory of the product CD-ROM. After installation, the readme.htm file is located in the root directory of the Partner Agreement Manager installation.
- The StartHere.htm file. This file contains links to the Partner Agreement Manager readme.htm file and the *Partner Agreement Manager Installation Guide*. Before installation, the StartHere.htm file is located in the root directory of the product CD-ROM. After installation, the StartHere.htm file is located in the root directory of the Partner Agreement Manager installation.
- The *Partner Agreement Manager Installation Guide*, form number GC34-5964-02, which describes how to install Partner Agreement Manager.

- The *Partner Agreement Manager Administrator's Guide*, form number BIAAAB02, which describes how to set up, configure, and administer Partner Agreement Manager after you install it.
- The *Partner Agreement Manager User's Guide*, form number BIAAAC02, which describes how to start a Partner Agreement Manager session, design public and private processes, define element definition sets, create business objects, and manage process distribution.
- The *Partner Agreement Manager Adapter Developer's Guide*, form number BIAAAD02, which describes how to develop and administer adapters using the Partner Agreement Manager Adapter Development Environment.
- The *Partner Agreement Manager Script Developer's Guide*, form number BIAAAE02, which describes how to write scripts used in Partner Agreement Manager private processes and elsewhere.
- The *Partner Agreement Manager External API Guide*, form number BIAAAF02, which describes principles behind the Partner Agreement Manager External API. See also, the Javadoc for the External API, which is installed in the Partner Agreement Manager Docs folder.
- The *Partner Agreement Manager Adapters for MQSeries User's Guide*, form number BIAAAG02, which describes how to install, configure, and run the Partner Agreement Manager Adapters for MQSeries.
- The *Partner Agreement View User's Guide*, form number GC34-5965-02, which describes how to install, configure, and use Partner Agreement View.

Summary of Changes

This edition includes these changes since the previous, first, edition:

- *External APIs. Partner Agreement Manager* 2.2 provides added flexibility to external applications through additional APIs. These APIs allow third-party applications to take advantage of the Partner Agreement Manager partner management and process engine through programmatic access. The API is distributed as a set of Java classes that the external application can import. Communication between the API classes and the Process Server is through RMI, but in the future can be swapped out for HTTP or SOAP. Specifically, APIs have been added to the following functional areas:
  - Session Service API
  - Admin Service API
  - Document Service API
  - Partner Service API
  - Adapter Service API
  - Process Service API

- *LDAP Support. Partner Agreement Manager* 2.2 provides centralized user authentication and administration through an LDAP directory. Partner Agreement Manager can retrieve user information—such as name, e-mail address, phone, and fax—stored in an LDAP directory. Updating this information is done in a single place, through the LDAP management tool. Users are authenticated through the same directory, giving them single-sign-on capabilities across enterprise applications.
- *Double-byte character sets (DBCS) and National Language Support (NLS).* Double-byte character sets are now supported in Partner Agreement Manager 2.2. Double-byte and multibyte data can be transferred and operated on in business objects and adapters. NLS lets Partner Agreement Manager display user interface text in other languages.
- *Improved XML Support.* The Partner Agreement Manager 2.2 engine fundamentally changes the way it interacts with business objects by replacing proprietary parsers with a third-party parser. This simplifies support of DTD 1.0 and the support of XML schemas when the standard is finalized.

  The Business Object and Script API have been extended with new classes and methods. The new classes and methods let you work with business objects as W3C Documents.
- *Adapter Asynchronous Callback.* An additional Adapter API allows adapters to be more efficient with long-running adapter operations. The Asynchronous Callback method tells the Adapter Server that an operation will be long-running, that system resources should be freed while the adapter waits for a response from the end system, and that another method will be called when the response arrives. The Asynchronous Callback method frees the adapter developer from using the request-retry method that makes the Adapter Server responsible for polling the end system for the response.
- *Script API Changes.* The script API now provides access to the PartnerGroupContext and the Public and Private Process Contexts. Through these contexts, you can get information such as partner group binding, a reference to the process, inputs to the process (which contain a reference to the sender, the ID of the sending node, and the variable name), and unique node and loop IDs.

- *Certificate Support. Partner Agreement Manager* 2.2 is able to request and import certificates from certificate authorities like VeriSign. This lets organizations use their existing certificate, or request a new one if their partners do not accept self-signed certificates. Partner Agreement Manager 1.1 supported only self-signed certificates.
- *Outbound Proxy Support. Partner Agreement Manager* 2.2 channels that use HTTP communication can work with outbound proxies that use authentication. Outbound proxy authentication is used within *internal* networks to ensure that only people and applications that are authenticated may communicate with an *external* network. Authentication in the outbound proxy is done with a standard user name and password combination. You can turn on the outbound proxy feature after installation. Thereafter, all outbound HTTP communication uses the same user name and password combination for the proxy.

NOTE: Note that this feature is only used by channels using HTTP communication; it does not apply to channels that use the built-in Partner Agreement Manager proxy.

CHAPTER

1

Introducing adapters

Read this chapter for an overview of WebSphere Partner Agreement Manager adapter technology and the environment that provides Partner Agreement Manager for developing adapters.

This chapter includes these sections:

- *Adapter overview* on page 2.
- *Adapter architecture* on page 4.
- *About the Adapter Development Environment* on page 8.

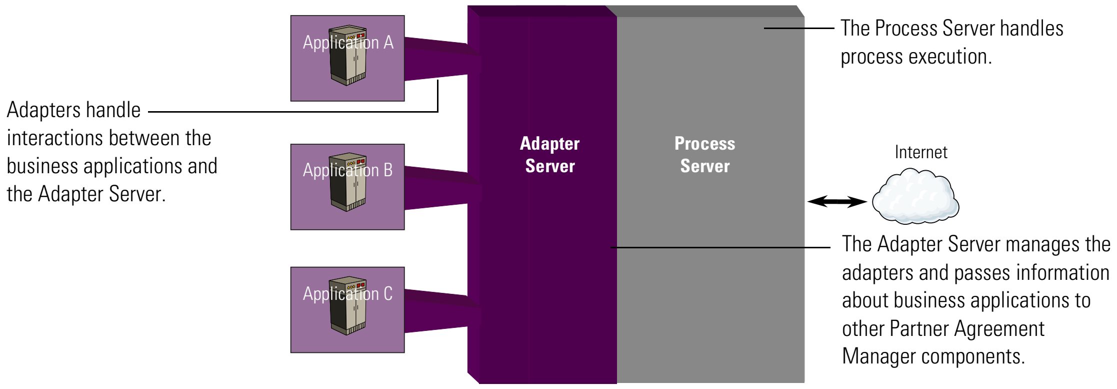
Note: We assume that you have read the *Partner Agreement Manager User's Guide*, the *Partner Agreement Manager Administrator's Guide,* and the *Partner Agreement Manager Installation Guide*. Online versions of these documents are available in your WebSphere/`PAM/Docs` directory. These documents help you understand Partner Agreement Manager Extension actions, events, and business objects, which are essential to understanding adapters. We also assume that you have experience in Java™ development.

Adapter overview

Within an organization, legacy and other enterprise systems play a key role in the company's ability to function and compete. Therefore, an effective business-to-business integration solution must integrate seamlessly with each partner's existing enterprise systems and provide extensibility to new and emerging technologies. Partner Agreement Manager provides these capabilities to enable end-to-end process execution using the heterogeneous systems of each partner.

Partner Agreement Manager's unique adapter technology serves as the bridge between Partner Agreement Manager processes and specific business applications interfaces. Adapters mask the complexity and diversity of interface technologies by wrapping each application interface in a common abstraction. This gives private process authors a uniform way to access the capabilities of a variety of external applications. Adapters also provide a uniform means of propagating events out of business applications so these events can be used to trigger Partner Agreement Manager processes.

Within the Partner Agreement Manager installation, the Adapter Server and adapters work together to manage interactions between other Partner Agreement Manager components and your business applications.

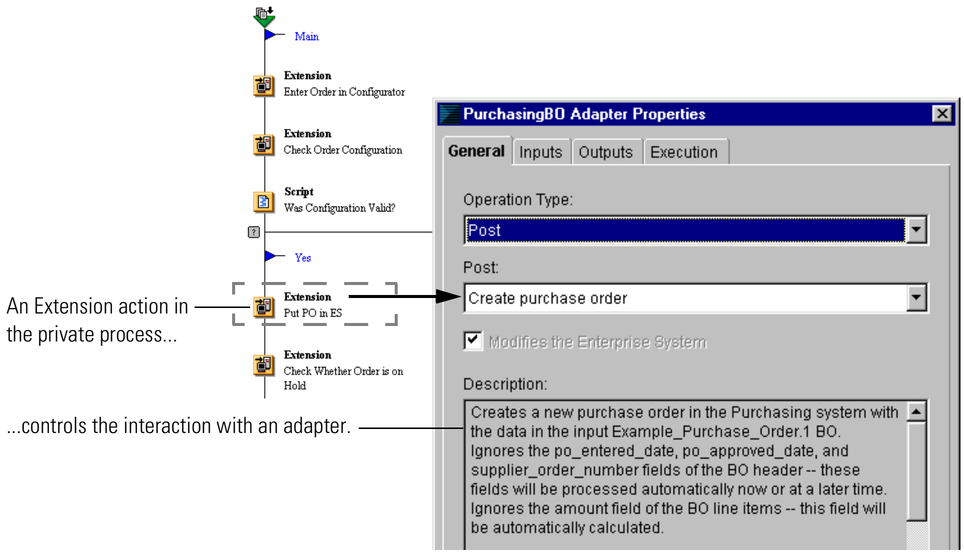

Application A
Application B
Application C
Adapters handle interactions between the business applications and the Adapter Server.
Adapter Server
Process Server
The Process Server handles process execution.
Internet
The Adapter Server manages the adapters and passes information about business applications to other Partner Agreement Manager components.

Adapters handle interactions between the external business applications and the Adapter Server. The Adapter Server, in turn, manages the adapters and serves as a conduit for configuration and other information about external business applications. The Adapter Server also ensures consistent handling of state management and audit capture within a process.

Interactions between adapters and Partner Agreement Manager processes are controlled by:

- private process Extension actions, which allow process designers to embed an unlimited number of external business application interactions into a private process definition, and
- events, which let Partner Agreement Manager users start public processes when an event is received.

Extension action properties determine the target business application, the type of interaction, and the type of information that is passed between the process and the business application.


Main
Extension
Enter Order in Configurator
Extension
Check Order Configuration
Script
Was Configuration Valid?
Yes
Extension
Put PO in ES
Extension
Check Whether Order is on Hold
An Extension action in the private process...
...controls the interaction with an adapter.
PurchasingBO Adapter Properties
General
Inputs
Outputs
Execution

Operation Type: Post

Post: Create purchase order

☑ Modifies the Enterprise System

Description:

Creates a new purchase order in the Purchasing system with the data in the input Example_Purchase_Order.1 BO. Ignores the po_entered_date, po_approved_date, and supplier_order_number fields of the BO header -- these fields will be processed automatically now or at a later time. Ignores the amount field of the BO line items -- this field will be automatically calculated.

Adapters perform two main functions within Partner Agreement Manager. They allow:

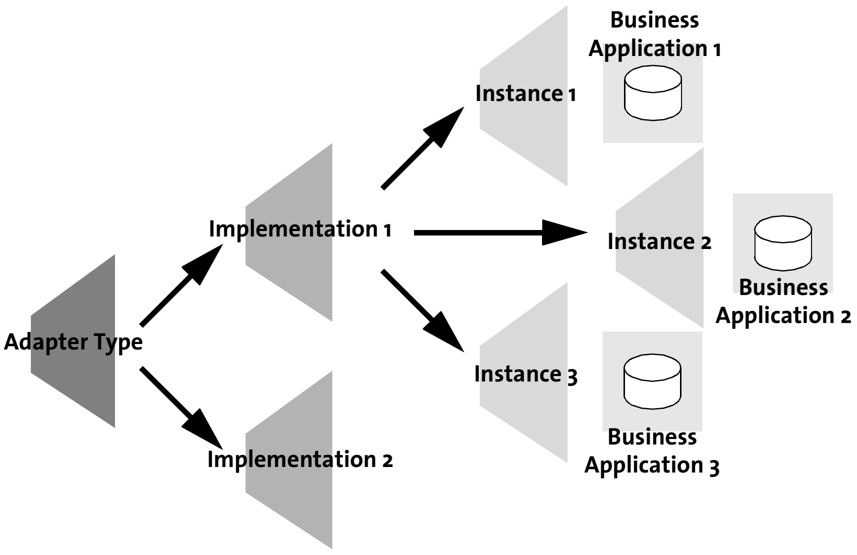
- Partner Agreement Manager private processes to interact with external business applications while a process is running. This interaction can take the form of obtaining information from a particular business application or passing information to it.
- Partner Agreement Manager to start public processes automatically based on events that occur in external business applications.

Adapter architecture

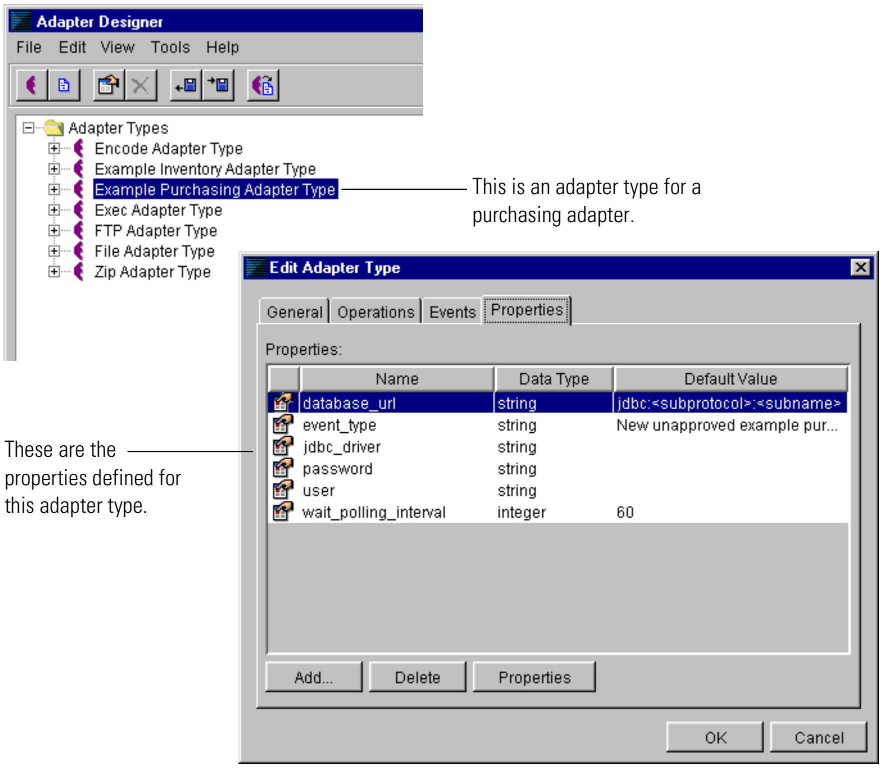
On an architectural level, an adapter is made of an adapter type, an adapter implementation declaration, and an adapter instance. A single adapter type can have several implementations, and each implementation can have several instances. For example, you might have separate implementations that use different persistence mechanisms, such as a file-based or database implementation, or that use different tax calculation methods, such as one from a library or one that does the calculations in the adapter code.

Another way to use implementations is for different versions—every time you make a major functional change, you can add another implementation, so you can always go back to the previous implementation.

Different instances of the same implementation are also useful. A common approach is to use different adapter instances for test and production. In the implementation, you can define properties such as database name, logon and password. In the test instance, you can point these to the test database, and debug without altering any of the production data. In the production instance, the properties point to the production database.

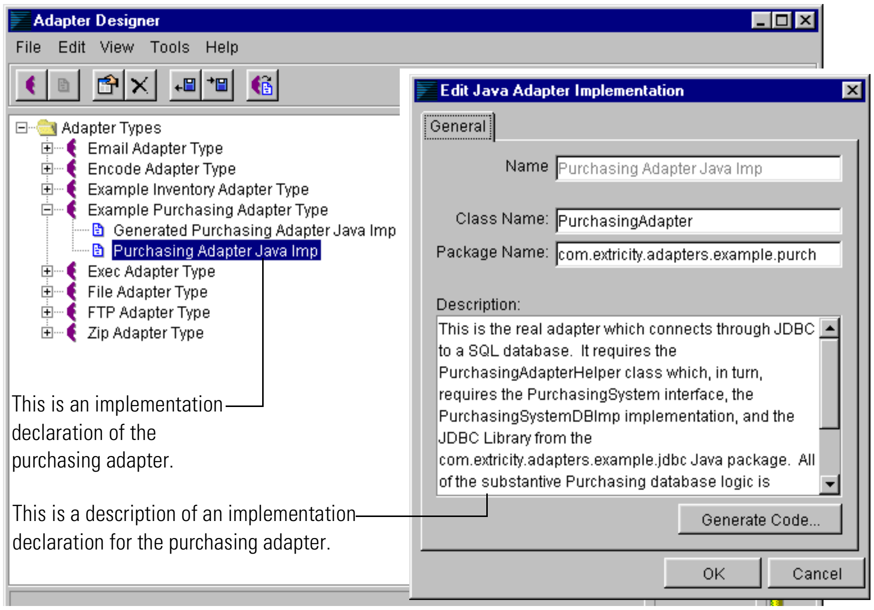


The adapter type, which is stored in XML format for import and export, contains adapter information, including what operations, events, and properties the adapter has. You specify this information in the Create Adapter Type dialog box in the Adapter Designer. For example, these are the properties defined in the adapter type for the example purchasing adapter.



Name	Data Type	Default Value
database_url	string	jdbc:<subprotocol>:<subname>
event_type	string	New unapproved example pur...
jdbc_driver	string	
password	string	
user	string	
wait_polling_interval	integer	60

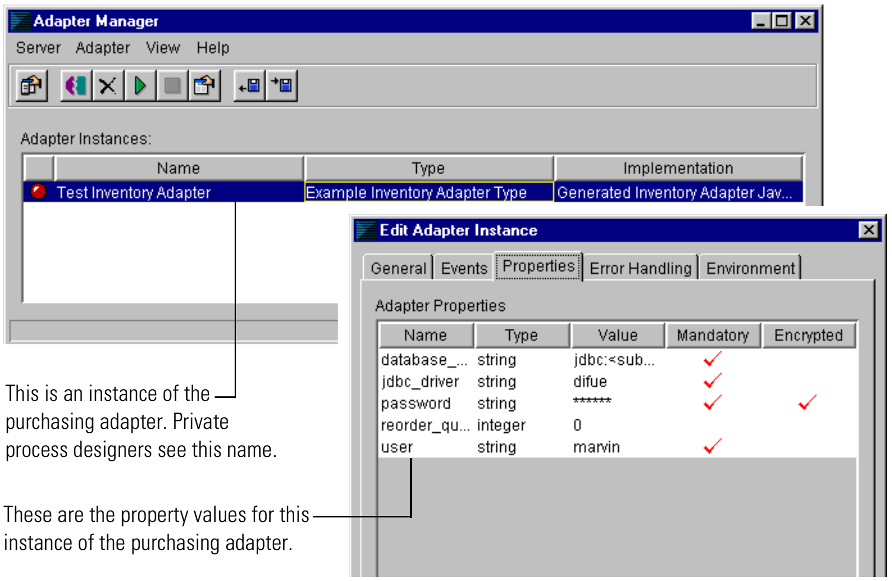
The adapter implementation declaration contains the Java source file name, class name, package, and directory path. The Java source file contains the application logic to communicate with a specific business application through that application's interface. The implementation declaration is imported and exported in XML. For example, following is an implementation declaration for the example purchasing adapter, subordinate to the adapter type. You specify this information in the Create Java Adapter Implementation dialog box in the Adapter Designer.



Adapter Designer
File Edit View Tools Help
Adapter Types
Email Adapter Type
Encode Adapter Type
Example Inventory Adapter Type
Example Purchasing Adapter Type
Generated Purchasing Adapter Java Imp
Purchasing Adapter Java Imp
Exec Adapter Type
File Adapter Type
FTP Adapter Type
Zip Adapter Type
This is an implementation declaration of the purchasing adapter.
This is a description of an implementation declaration for the purchasing adapter.
Edit Java Adapter Implementation
General
Generate Code...
OK
Cancel

Name: Purchasing Adapter Java Imp
Class Name: PurchasingAdapter
Package Name: com.extricity.adapters.example.purch
Description:
This is the real adapter which connects through JDBC to a SQL database. It requires the PurchasingAdapterHelper class which, in turn, requires the PurchasingSystem interface, the PurchasingSystemDBImp implementation, and the JDBC Library from the com.extricity.adapters.example.jdbc Java package. All of the substantive Purchasing database logic is

The adapter instance provides specific values for the properties declared in the implementation. For example, the following are the property values for an instance of the example purchasing adapter.



Name	Type	Implementation
Test Inventory Adapter	Example Inventory Adapter Type	Generated Inventory Adapter Jav...

Name	Type	Value	Mandatory	Encrypted
database_...	string	jdbc:<sub...	✓	
jdbc_driver	string	difue	✓	
password	string	******	✓	✓
reorder_qu...	integer	0		
user	string	marvin	✓	

This is an instance of the purchasing adapter. Private process designers see this name.

These are the property values for this instance of the purchasing adapter.

The adapter instance is also:

- where you set how often the adapter should poll for events.
- the name that private process designers specify when they use the adapter.
- the set of property values for the instance.
- the environment settings for the adapter instance at run time.

About the Adapter Development Environment

Partner Agreement Manager provides a comprehensive Adapter Development Environment (ADE) that gives you an intuitive framework for configuring, customizing, and developing adapters.

The ADE includes these components:

- The Adapter Designer, a graphical user interface for building custom adapters. See *About the Adapter Designer* on page 43.
- The Integration Wizard Framework. This framework is part of the Adapter Designer and provides services for integration wizards. It helps you create adapters with minimal effort. Using a graphical interface, a wizard lets you specify the details about the external system and the business objects you want generate. The wizard generates the business objects and the adapter. Depending on your system and requirements, the adapter might require some coding. Even for those systems that do require additional coding, an integration wizard can greatly reduce the amount of time it takes to develop an adapter. See *Using an integration wizard* on page 9 for a walk-through of the Flat File Integration Wizard.
- A set of example and utility adapters.

  Example adapters are fully annotated and functional adapters that demonstrate elements commonly found in an adapter. You can use example adapters as the basis for any custom adapters you want to build. The example adapters included in the Partner Agreement Manager ADE are used as illustrations throughout this guide.

  Utility adapters are pre-built adapters that handle connections to generic systems such as an FTP server, a mail server, or a file server. Process designers can select utility adapters for enterprise actions the same way they would select any other adapter. See *Using utility adapters* on page 83.

CHAPTER

2

USING AN INTEGRATION WIZARD

Read this chapter for information on configuring an integration wizard and a short tutorial on how to use the Flat File Integration Wizard to create an adapter for a flat file.

This chapter includes these sections:

About configuring integration wizards

Each integration wizard has server-side and client-side properties that provide information that the integration wizard uses when it creates adapters.

The server-side properties identify the integration wizard and show the location of the archive in which it is packaged. The only server-side property that you can set is the CLASSPATH, which specifies the path to the directory where adapter instances created by the integration wizard must look for any external classes that they require. If the integration wizard does not require external classes, the CLASSPATH can be blank. All other server-side properties are set by the installer.

The client-side properties are used by the integration wizard to connect to the target end system when generating adapters.

- Class name specifies the class that the integration wizard uses as its starting point. The name of the class is set by the installer and cannot be edited.
- CLASSPATH specifies the path to the directory on the client-side computer where the integration wizard needs to look for any external classes that it requires. If the integration wizard does not require external classes, the CLASSPATH can be blank.

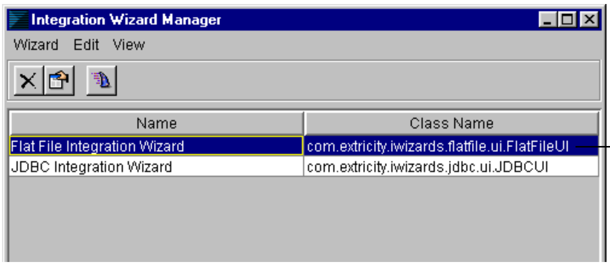
Configuring an integration wizard

You configure an integration wizard by setting its CLASSPATH properties.

To configure an integration wizard:

1 In the Adapter Designer, choose Integration Wizard Manager from the Tools menu.

The Integration Wizard Manager appears, displaying any integration wizards that have been installed.

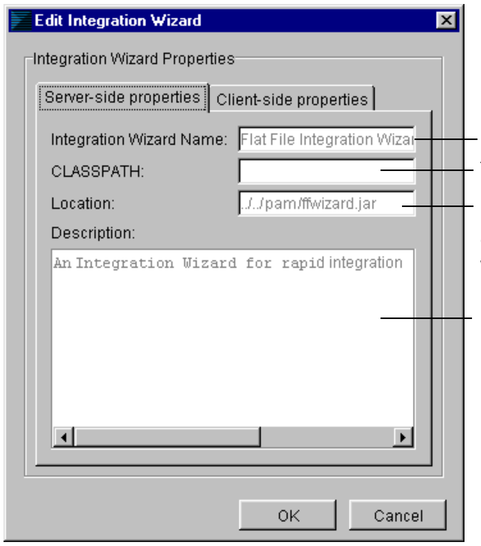


Name	Class Name
Flat File Integration Wizard	com.extricity.iwizards.flatfile.ui.FlatFileUI
JDBC Integration Wizard	com.extricity.iwizards.jdbc.ui.JDBCUI

Any integration wizards that have been installed appear here.

2 Double-click an integration wizard to edit its properties.

The Edit Integration Wizard dialog box appears. Most of the server-side properties are already set, but you might be required to supply a CLASSPATH if adapters generated by the integration wizard depend on external classes.



Integration Wizard Name: Flat File Integration Wizar

CLASSPATH:

Location: ../../pam/ffwizard.jar

Description:

An Integration Wizard for rapid integration

Shows the name of the integration wizard.

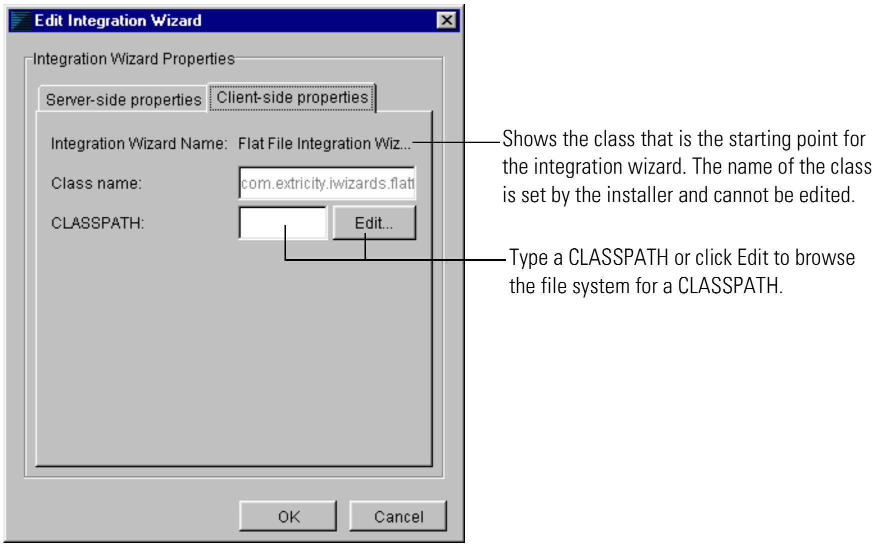
Type a CLASSPATH here.

Shows the location of the Java archive where the integration wizard package is.

Shows a description for the integration wizard.

3 Click the Client-side properties tab to Set the client-side CLASSPATH.

The Class name is already set, but you might be required to supply a CLASSPATH if the integration wizard requires external class files to allow it connect to adapter's the end system.


Edit Integration Wizard

Integration Wizard Properties

Server-side properties | Client-side properties

Integration Wizard Name: Flat File Integration Wiz...

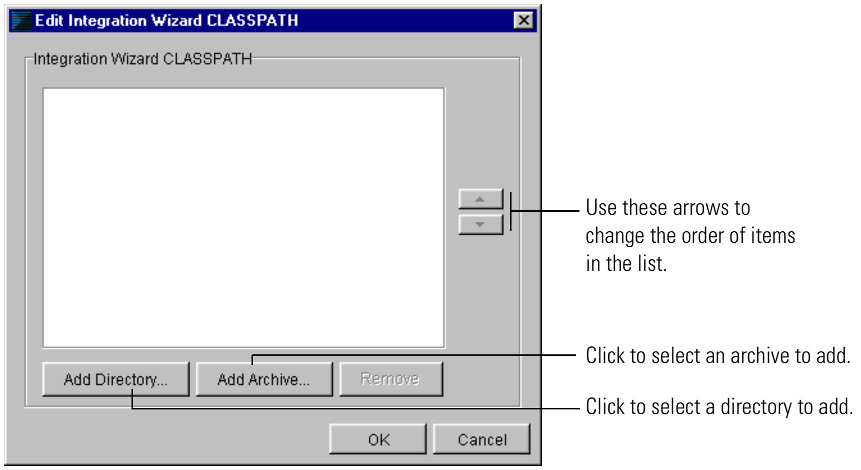
Class name: com.extricity.iwizards.flatt

CLASSPATH:

Edit...
OK
Cancel
Shows the class that is the starting point for the integration wizard. The name of the class is set by the installer and cannot be edited.
Type a CLASSPATH or click Edit to browse the file system for a CLASSPATH.

4 Type a CLASSPATH, or click the Edit button to compose the CLASSPATH.

The Edit Integration Wizard CLASSPATH dialog box appears. You can use it to build a CLASSPATH by browsing the file system on the client-side computer and selecting directories or archives. You can also type a CLASSPATH.



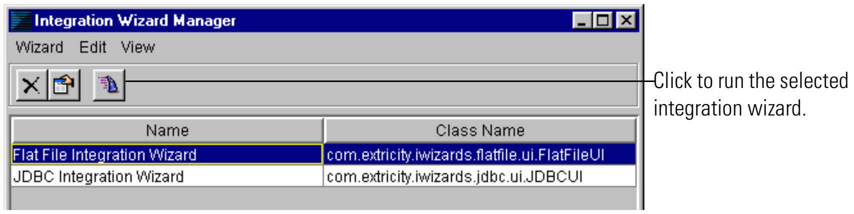
Edit Integration Wizard CLASSPATH
Integration Wizard CLASSPATH
Add Directory...
Add Archive...
Remove
OK
Cancel
Use these arrows to change the order of items in the list.
Click to select an archive to add.
Click to select a directory to add.

5 Click OK in the Edit Integration Wizard CLASSPATH dialog box and in the Edit Integration Wizard dialog box.

Managing integration wizards

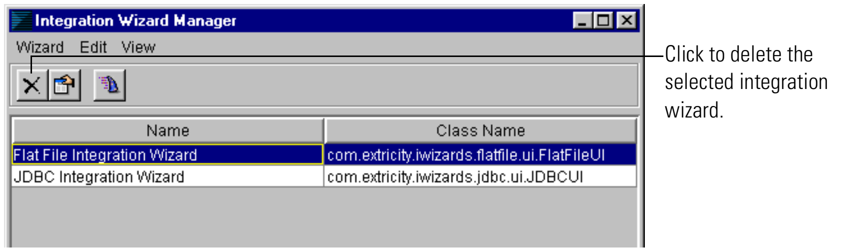
You can use the Integration Wizard Manager to run or delete an integration wizard.

- To run an integration wizard, select the wizard in the list and click the Run button.


Integration Wizard Manager
Wizard
Edit
View
Click to run the selected integration wizard.

Name	Class Name
Flat File Integration Wizard	com.extricity.iwizards.flatfile.ui.FlatFileUI
JDBC Integration Wizard	com.extricity.iwizards.jdbc.ui.JDBCUI

- To delete an integration wizard, select the wizard in the list and click the Delete button.


Integration Wizard Manager
Wizard
Edit
View
Click to delete the selected integration wizard.

Name	Class Name
Flat File Integration Wizard	com.extricity.iwizards.flatfile.ui.FlatFileUI
JDBC Integration Wizard	com.extricity.iwizards.jdbc.ui.JDBCUI

About the Flat File Integration Wizard

The end product of the Flat File Integration Wizard is a working adapter that can perform these basic functions with a flat data file:

- Read a data file and populate a business object.
- Write the business object data to a flat file.

A flat file adapter reads data from a flat file and uses that data to populate a business object. That business object can then be used by another adapter, or by Partner Agreement Manager processes just like any other business object. A flat file written out by a flat file adapter can be used as input for another business system or for archival purposes.

This tutorial takes you through the process of defining and creating an adapter for flat files.

The steps you'll go through to create a working flat file adapter are:

Step 1 On the machine running the Adapter Server, open the Process Manager window. Start the Flat File Integration Wizard.

Step 2 Choose the operations you want this adapter to perform.

Step 3 Define the input file format and fields.

Step 4 Specify the name and choose options for the business object.

Step 5 Specify the name for the adapter type, implementation, and instance.

Step 6 The Flat File Integration Wizard generates the business object, adapter type, adapter implementation, and adapter instance.

Before you begin, make sure you have installed the Process Manager, the Process Server, and the Adapter Server on the computer you'll be using.

Using the Flat File Integration Wizard

The Flat File Integration Wizard walks you through the process of making an adapter for flat files.

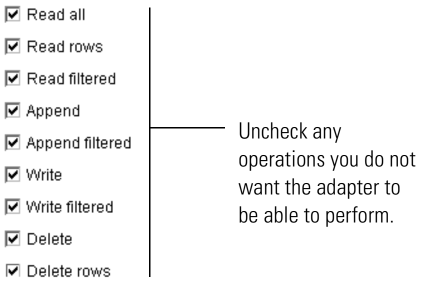
To make a flat file adapter:

1 In the Adapter Designer, choose Create Adapter Using Wizard from the File menu. Then choose Flat File Integration Wizard.

This starts the Flat File Integration Wizard. The Welcome window appears.

2 Click Next to start building your adapter for flat files.

The Select Adapter Operations window appears.



- ☑ Read all
- ☑ Read rows
- ☑ Read filtered
- ☑ Append
- ☑ Append filtered
- ☑ Write
- ☑ Write filtered
- ☑ Delete
- ☑ Delete rows

Uncheck any operations you do not want the adapter to be able to perform.

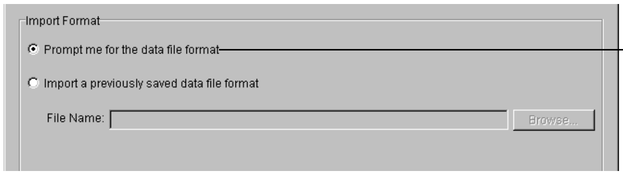
3 Set the operations you want the adapter to perform. Click Next to continue.

By default, all the operations are checked. The available options are:

Check this option	To
Read All	Read all the data from the input file and populate the business object.
Read Rows	Read rows from the input file and populate the business object.
Read Filtered	From the input file, read rows that satisfy the filter criteria. Specify the filter criteria in a Flat_File_Filter business object that you instantiate and populate in a script action before calling the adapter.
Append	Append all the data in the business object to the output file.

Check this option	To
Append Filtered	Append all the data that satisfies the filter criteria to the output file. Specify the filter criteria in a Flat_File_Filter business object that you instantiate and populate in a script action before calling the adapter.
Write	Write all the data in the business object to the output file you specify. If the file doesn't exist, a new file with that name is created. If a file already exists with that name, it is overwritten.
Write Filtered	Write data from the business object to the output file you specify. Only data that satisfies the filter criteria will be written. Specify the filter criteria in a Flat_File_Filter business object that you instantiate and populate in a script action before calling the adapter. If the file doesn't exist, a new file with that name is created. If a file already exists with that name, it is overwritten.
Delete	Delete the input file.
Delete Rows	Delete rows from the input file.
Delete Filtered	Delete rows from the input file. Only data that satisfies the filter criteria will be deleted. Specify the filter criteria in a Flat_File_Filter business object that you instantiate and populate in a script action before calling the adapter.
Update	Read rows from the input file that satisfy the filter criteria. Update the values in those rows with data from the Flat_File_Update business object. Specify the filter criteria and update values in a Flat_File_Update business object that you instantiated and populated in a script action before calling the adapter.
Get Count	Return the number of rows in the input file that satisfy the filter criteria. Specify the filter criteria in a Flat_File_Filter business object that you instantiate and populate in a script action before calling the adapter.

The Specify Data File Format Method window appears.


Import Format

- (●) Prompt me for the data file format
- ( ) Import a previously saved data file format

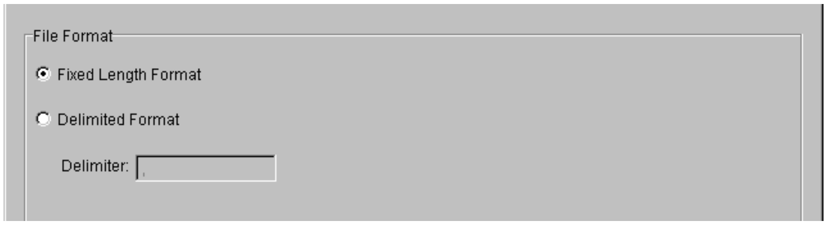
File Name:

Because you're specifying this data format for the first time, select this setting.

4 Choose Prompt Me for the Data Format. Click Next.

Because you're specifying the data format for this input file for the first time, select Prompt Me for the File Format. When you've specified the file format, you can save the specification and import it the next time you want to create an adapter for this data file format.

The Specify Data Format window appears.


File Format

- (●) Fixed Length Format
- ( ) Delimited Format

Delimiter: ,

5 Choose Fixed Length Format. Click Next to continue.

Each input data file has information for exactly one business object. This file can be either fixed length or delimited by some character, usually a comma. A fixed format file always has the same field beginning in the same place on each line. Delimited text fields are run one after the other, with no spaces padding out the field length. The end of each field is indicated by a delimiter such as a comma.

If your data consisted of an ID and a name, a fixed length format would look like this:

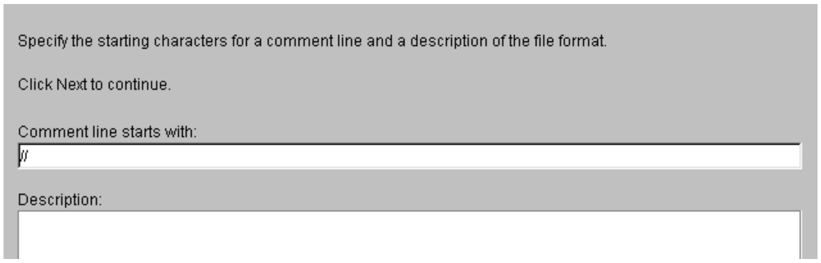
```
17          Al Frahm
```

A delimited format would look like this:

```
17, Al Frahm
```

This example uses a fixed length format. The screens for a delimited format are slightly different.

The Specify Comment Delimiter window appears.


Specify the starting characters for a comment line and a description of the file format.

Click Next to continue.

Comment line starts with:

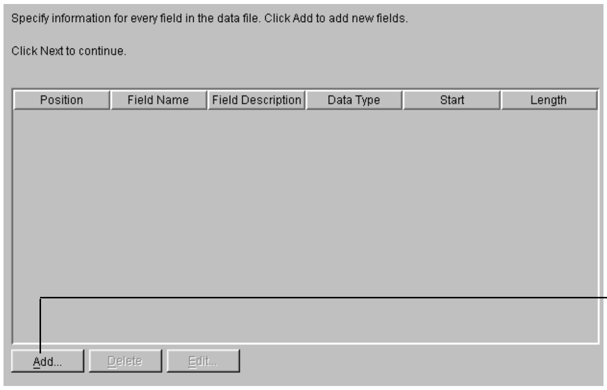
//

Description:

6 Specify the comment delimiters. Enter a description if you want, and click Next to continue.

Input data files frequently have comment lines that describe the file. Indicate the comment delimiters so the adapter does not process these comment lines as data. You can also add some description of the file format for your organization's internal documentation.

The Specify Field Details window appears.


Specify information for every field in the data file. Click Add to add new fields.

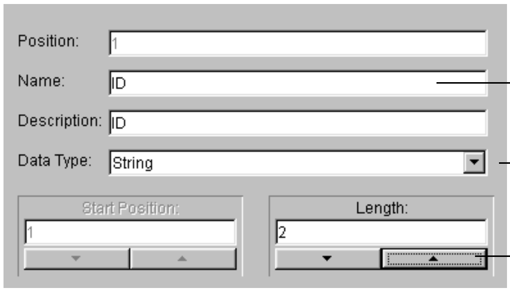
Click Next to continue.

Position	Field Name	Field Description	Data Type	Start	Length

Click Add to add fields to the specification.

7 Click Add to add information about the first field in the data file.

The Specify Field Information window appears.


Position: 1

Name: ID

Description: ID

Data Type: String

Start Position: 1

Length: 2

Give the field a descriptive name.

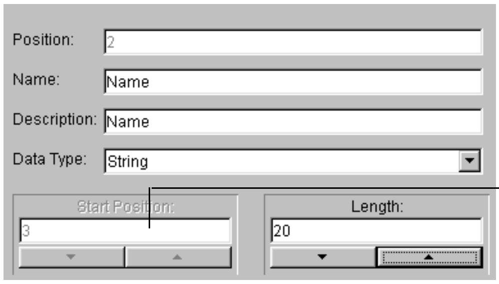
Choose one of the available data types for this field.

Use the roller arrows to set the length of this field.

8 Enter the details about the first field in the row. Click OK.

If your input file had just 2 fields: an ID field and a name field, the first field would be named ID, be of type String, and have a length of 2.

9 Click Add again in the Specify Field Details window to enter the details of the next data field.


Position: 2

Name: Name

Description: Name

Data Type: String

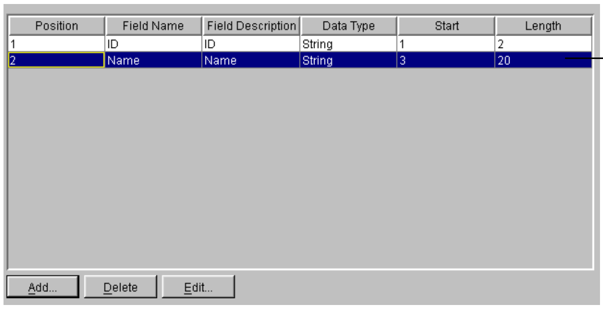
Start Position: 3

Length: 20

Note that the start position is already calculated for you.

Using the same example, your second field would be named Name, be of type String and have a length of 20. Type this and click OK to continue.

Your field information looks like this:

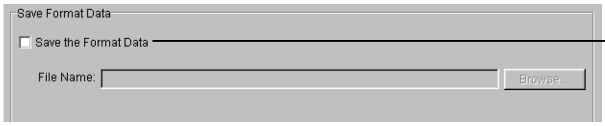


Position	Field Name	Field Description	Data Type	Start	Length
1	ID	ID	String	1	2
2	Name	Name	String	3	20

You can change field order by dragging a field to a different position in the list.

10 Click Next when you finish adding the field information.

The Save Format Data window appears.



Save Format Data

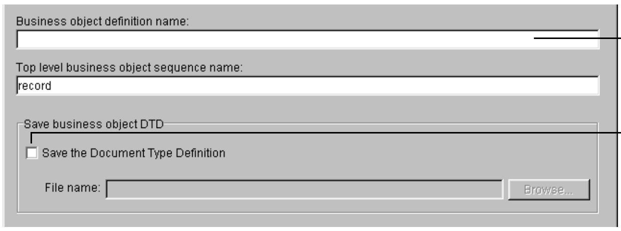
☐ Save the Format Data

File Name:

Turn this setting on to save the format you specified.

11 Turn the Save the Format Data setting on to save the format specification you just made. Click Next to continue.

The Specify Options for Business Object window appears.



Business object definition name:

Top level business object sequence name: record

Save business object DTD

☐ Save the Document Type Definition

File name:

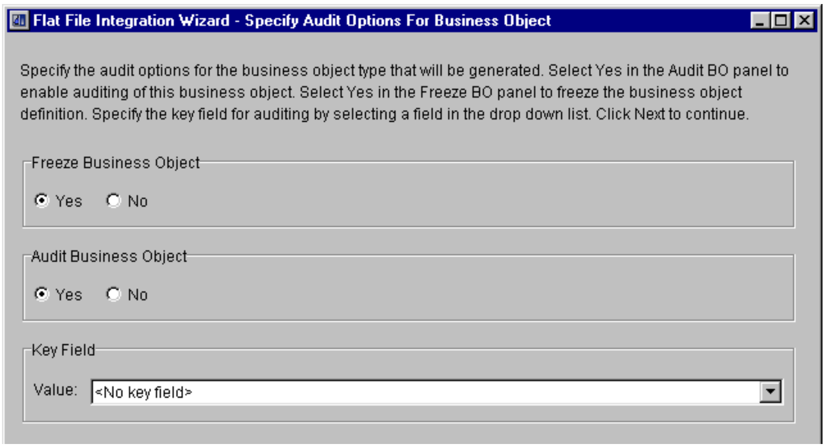
Give the business object a descriptive name.

Check this box and enter a file name to save this business object specification to a file.

12 Set options for the business object. Click Next to continue.

- Enter a name for the business object definition.
- Enter the name of the top level business object sequence.
- If you want to save this business object definition, turn on the Save the Document Type Definition setting and specify a file name.

The Specify Audit Options For Business Object window appears.



Flat File Integration Wizard - Specify Audit Options For Business Object

Specify the audit options for the business object type that will be generated. Select Yes in the Audit BO panel to enable auditing of this business object. Select Yes in the Freeze BO panel to freeze the business object definition. Specify the key field for auditing by selecting a field in the drop down list. Click Next to continue.

Freeze Business Object: [✓ Yes] No

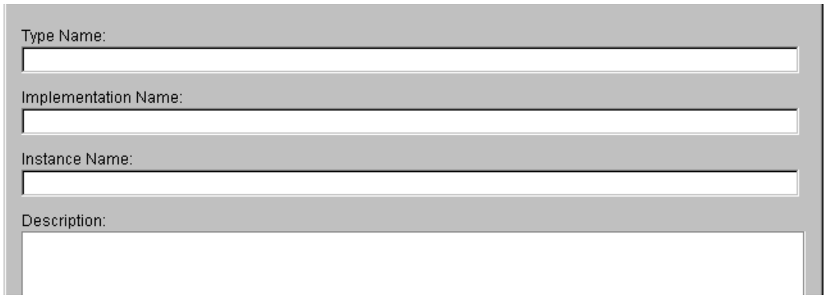
Audit Business Object: [✓ Yes] No

Key Field

Value: <No key field>

13 Specify whether to freeze the business object and whether to turn auditing on. If you need one, set a key field for the business object. Click Next to continue.

The Specify Adapter Options window appears.


Type Name:

Implementation Name:

Instance Name:

Description:

14 Specify the adapter options. Click Finish to generate the adapter.

- Type the name of the adapter type. This appears in the adapter types lists in both the Adapter Designer and the Adapter Manager.
- Type the implementation name. This appears in the Adapter Manager when you add an adapter instance.
- Type the instance name. This appears in the Adapter Manager.

When you click Finish, the integration wizard generates the adapter type, implementation, instance, and business objects.

15 When the process is complete, click OK.

If you go to the Adapter Manager, you see a running instance of this adapter.

CHAPTER 3

Designing Adapters

Read this chapter for information about designing custom adapters.

This chapter includes these sections:

About developing adapters

An adapter enables interaction between the Partner Agreement Manager process and a business system. This business system can be anything from a simple spreadsheet application to a database to a sophisticated business application such as an enterprise resource planning system.

These are the types of adapters you can use:

- Pre-built adapters that are distributed with Partner Agreement Manager. The utility adapters are a good example of pre-built adapters. They require no coding whatsoever on your part. To make these adapters available to process designers, you need only add adapter instances. For more on the utility adapters, see *Using utility adapters* on page 83. Adding adapter instances is described in the *Partner Agreement Manager Administrator's Guide*.
- Adapters built with an integration wizard. Integration wizards guide you through the process of building an adapter for a particular end system or application. The wizard walks you through the options and builds the adapter for you. At the end of the process, you have a running adapter. No coding is necessary. A good example of a wizard is the Flat File Integration Wizard, described in *Using an integration wizard* on page 9.
- Custom adapters. This last type of adapter is one that you code to meet the needs of your particular end system and business requirements. It is this type of adapter that requires design and coding.

The steps of designing an application are the same no matter what the business system is. However, the success of a given adapter depends on the adapter developer's in-depth knowledge of the business system's functionality, structure, and technical workings. You can't develop a good adapter without knowing a great deal about the business system you're writing the adapter for. This is especially true of custom adapters.

Developing an adapter is a multi-step process. The steps are:

- Designing the adapter
- Defining the adapter type
- Creating an adapter implementation
- Adding any additional code
- Creating an adapter instance
- Debugging and testing

Designing the adapter

The first step in any software design is to have a clear idea of what you need the software to do. You need to negotiate with the process designer to determine what information the adapter receives and what it is expected to be returned.

In general, it's a good idea to restrict the adapter's functionality to connecting with the business system, writing data to it and reading data from it. Other tasks such as data mapping are best handled outside the adapter, in the private process. This compartmentalization of functionality not only makes the adapter easier to write and debug, but it also vastly increases the possibility for reusable code.

An adapter that has been designed and written with reusability in mind can be used with many different private processes. You can have several instances of such an adapter, setting the properties that distinguish each instance at deployment time. This is good design. For example, you write an adapter to connect to a database and to read data from it. You can use one property to point to the correct database, another to hold the user name to log in and yet another to hold the password.

Error handling

As you design the methods of the adapter, you need to also correctly and robustly handle any error conditions that might arise. Where possible, the adapter must do at least a first-level error handling and recovery. Second level recovery can be handled by the Adapter Server. Returning to the database example, you have written code in the adapter's startup() method to try to connect to the database. If this connection attempt fails, the adapter must try to recover, in case the failure was due to a dropped network packet or some other momentary problem. You can do this by throwing an EndSystemNotAvailableException when the connection attempt fails. This exception will be caught by the Adapter Server, which then suspends the adapter and attempt to restart it. When you generate and install the adapter instance, you can set the properties that determine how the Adapter Server handles recovery when the end system is not available. See the *Partner Agreement Manager Administrator's Guide* for more on setting these properties.

You need to account for other errors as well by using either of the following options:

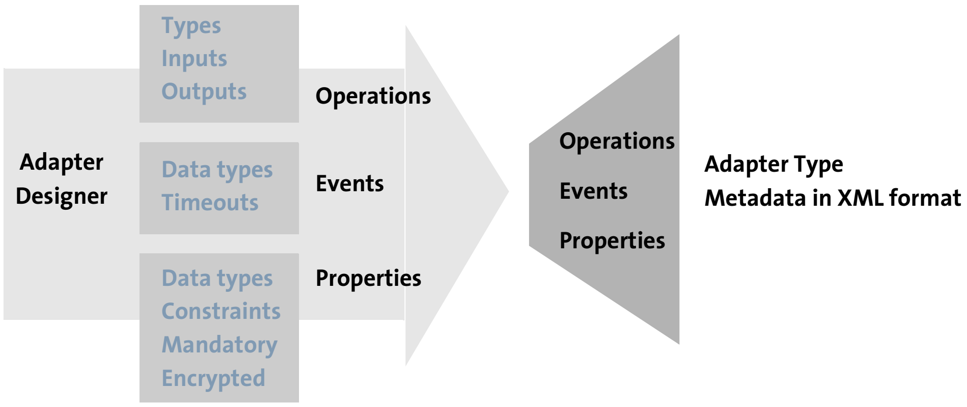
- Throw an ISException or let a run-time error propagate back to the Adapter Server. This results in a Private Action Error, which causes an error in the process.
- Catch all exceptions and use a StatusBO to give the private process a chance to handle the error at the private process level.

Clearly, catching exceptions where possible is the best option. However, you need to be careful to catch only the exceptions you know how to handle. Once you've caught the exception and passed the information on to the private process via a StatusBO, the private process must do everything possible to robustly handle the error. If the private process fails and escalates the error to the public process and the public process therefore fails, your partner(s) are notified. Doing this for a software or systems error that you can fix is like notifying the executive staff every time your workstation crashes.

Defining the adapter type

An adapter type serves as the blueprint for the implemented adapter: it defines the operations the adapter performs, the events the adapter passes from the business system to the Process Server, and the properties the adapter uses. It has a unique name to help you identify adapters that use the type.

You use the Partner Agreement Manager Adapter Designer to create the adapter type. Each part of the adapter type has its own set of required information that defines it.



Developing an adapter type is likely to be an iterative process as you refine the definitions for operations and events, add or remove them, or revise the definitions for properties. As you work, you can revise an existing adapter type, or use it as the basis for creating a new adapter type. To revise an existing adapter type, first make sure that all instances using the type are stopped. For more information, see *Defining a new adapter type* on page 44.

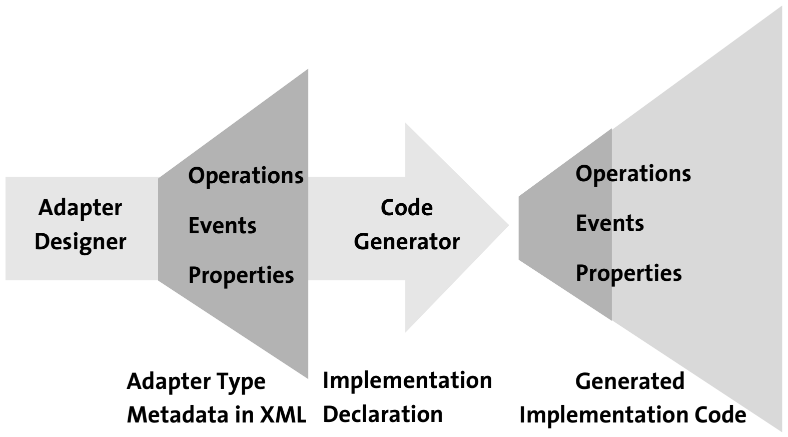
Creating an adapter implementation

When you're satisfied that the adapter type is complete, the next step is to create an implementation of the adapter type. An adapter implementation is a Java source file that is generated by the Adapter Designer. It contains Java code that provides the run-time implementation of the operations, events, and properties specified in the adapter type.

First, you create an implementation declaration in the Adapter Designer: you specify the Java source file name, class, and package names; a description; and a location for the generated files. After the Adapter Designer creates a Java file based on the information in the implementation declaration and the adapter type, you can edit it as needed and compile it.

The generated code contains placeholders for operations and event generation, and definitions for properties you specified. You must add your own Java code to fill in the placeholders with functionality that is specific to your external business system. The generated code also contains a method that loads the property values at run time.

For example, an implementation of an inventory adapter type might be one that contains code to get and post inventory data.


Adapter Designer
Operations
Events
Properties
Code Generator
Operations
Events
Properties
Adapter Type Metadata in XML
Implementation Declaration
Generated Implementation Code

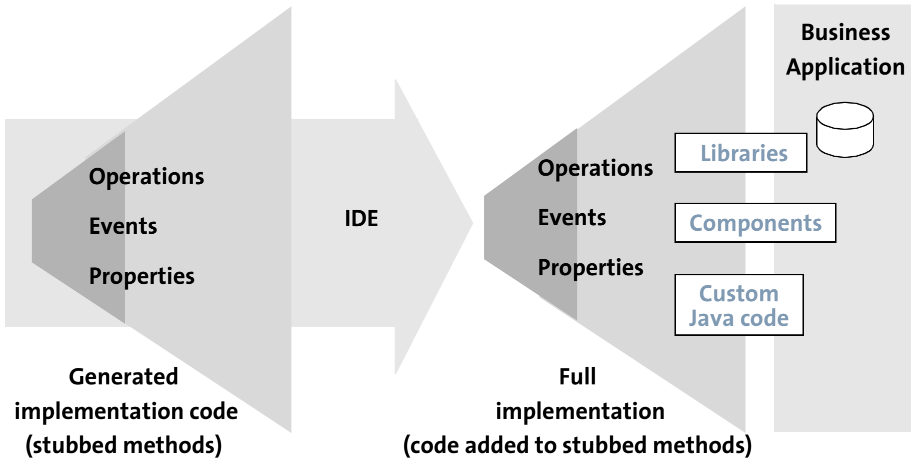
A single adapter type can have more than one implementation. For example, you might want to create one implementation for an in-memory test adapter, and another implementation—the real production version—that accesses a database. Or, you might want to create different implementations for different versions of the production adapter.

For more information on using the Adapter Designer to specify implementation information and to generate the code, see *Generating code for an adapter implementation* on page 63.

Note: Moving from adapter type to implementation is a one-way process. If you revise the adapter type definition after you generate an implementation, you must generate a new implementation to include those changes. Then merge in your custom code. The easiest way to do this is to create a helper class so that the effort of merging code is minimal.

Coding your adapter

The next step in the evolution of an adapter takes place in the integrated development environment (IDE) of your choice—for example, VisualAge® for Java—or a text editor. Here, you add Java code to the generated code for any application logic that is needed.



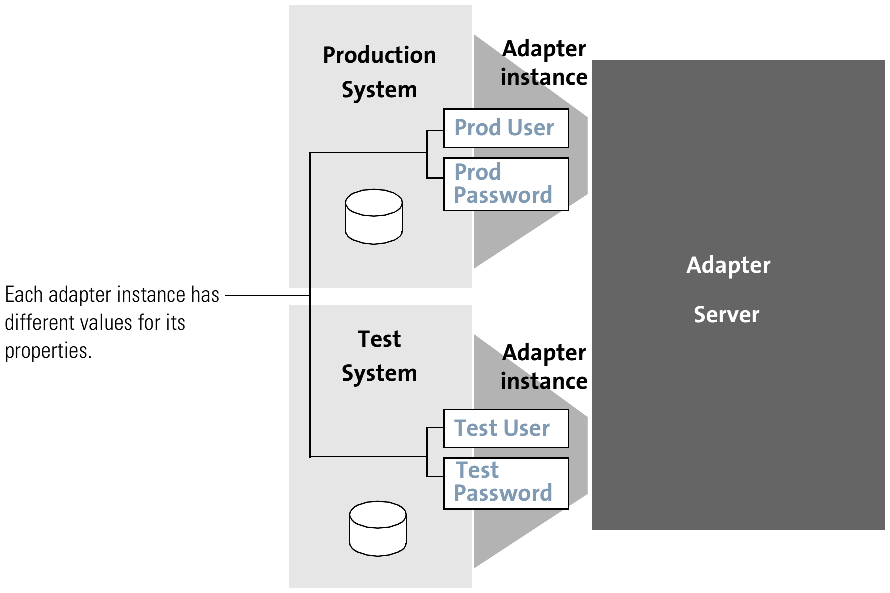
For example, your implementation can use function call APIs, document/ messaging APIs, and object APIs, as well as other interfaces. It can also emulate user sessions.

After you add code, you continue to work in your IDE of choice to compile the adapter. For more information, see *Using adapter API methods* on page 103, *Using Business Object API methods* on page 191, and *Adding custom code to adapters* on page 247. As part of the testing process, you use the Adapter Manager to create adapter instances that can be used in private process Extension actions, as described next.

Creating an adapter instance

An adapter instance is a run-able implementation that has been added in the Adapter Manager and has values for the mandatory properties defined in the adapter type. Partner Agreement Manager process developers use adapter instances when they include Extension actions in private processes. In addition, Partner Agreement Manager users can associate events created by an adapter with a public process, so when the event is received it starts the associated public process.

An example of an inventory adapter instance is one that connects with a specific database using the appropriate user name and password. You can create as many instances for an adapter as you need. For example, if you have more than one production inventory database, you can create an adapter instance for each.



For more information, see the *Partner Agreement Manager Administrator's Guide*.

Debugging and testing your adapter

After you've compiled your adapter and created an instance, you debug it. You must have an adapter instance to debug and test the adapter.

Planning an adapter—questions to consider

When you design an adapter, you develop a plan that identifies the business functions the adapter must perform and lays out a strategy for development. As you plan the adapter, here are some design questions to consider. The answers help you to construct an adapter that remains useful throughout the entire life cycle of the Partner Agreement Manager implementation.

What functions does the adapter perform?

The first step in planning an adapter is to determine what business requirements the adapter must fulfill and what functions it must perform. What system does it connect to? What types of interactions are required? What business objects are required?

Once you've established these basics, you can look at the integration problem to be solved and determine if it decomposes into technical and functional components. If so, you might want to divide the two tasks into different components.

The technical components can handle direct interaction with an external application. This might entail wrapping with Java some system interface in another programming language. There might be nothing related to Partner Agreement Manager in this part of the adapter.

The functional—business—components can then handle the semantic mapping between application data and Partner Agreement Manager data in the form of business objects and events. This part of the adapter can take care of reading inputs, using the right technical components, and then writing the outputs.

What is the starting point?

Once you've identified the functions to be performed, you can decide on a starting point. Choices include reusing an existing adapter—for example, an example adapter—or creating an entirely custom adapter. In addition, a utility adapter might provide all of the functions you need.

If an adapter module that connects to your business system is available, it becomes the most promising starting point. Adapter modules are designed to provide basic connectivity and a sample set of interactions with the target system. Using an adapter module makes it unnecessary for you to become an expert with the target system vendor's interface model. Instead, you can focus on implementing business interactions and leave the connectivity issues to the adapter module.

How much code is reusable?

Before you begin developing the adapter, it's a good idea to determine if there is integration work that would be reusable in other adapters to this enterprise system. Are there generic technical interfaces that all functional modules access? For a given adapter solution, can you see anyone reusing your underlying technical components without using the rest of the adapter? can other adapters reuse debugging or exception handling methods? can other adapters for the same system reuse your underlying code for establishing a connection, reading from the data dictionary, and so forth? If so, you might want to consider structuring the adapter in a way that provides the most reusability for your code.

The two levels of adapter code reuse are:

- reuse of a complete adapter in a different process.
- reuse of part of an adapter in another adapter.

The reuse of a complete adapter is ideal, necessitating no additional coding. Designing adapters to do the minimum necessary work maximizes this sort of reuse. Generally putting information into and getting information from the business system is all most adapters need to do. The manipulation of data and business objects can be done outside the adapter with scripts and mapping. For example, you might not want to post or get a partner business object directly into or from your enterprise systems. Instead, you can construct a business object that reflects your enterprise system and map that to the shared business object in the private process. This facilitates the reuse of this adapter for another process, perhaps with another partner.

If reusing the entire adapter isn't possible, you might find that a multi-level adapter design is best. This allows you to reuse portions of an existing adapter. In this scenario, to reuse code you might have to write several adapters for different interface points of the business system.

If, on the other hand, reusability is not an issue, you might find that creating a single-level adapter simplifies the task when you are building a custom adapter.

How much abstraction is possible?

In designing an adapter, you want to achieve as much abstraction as possible. A level of abstraction serves to insulate the private process from the system interface, so that changes in the system interface don't impact adapter operations.

To achieve this goal, consider whether the underlying technical components isolate the functional components of the adapter from changes in the business application. For example, in a database integration problem, can you change the table structure or a column format without changing the adapter?

About adapter type information

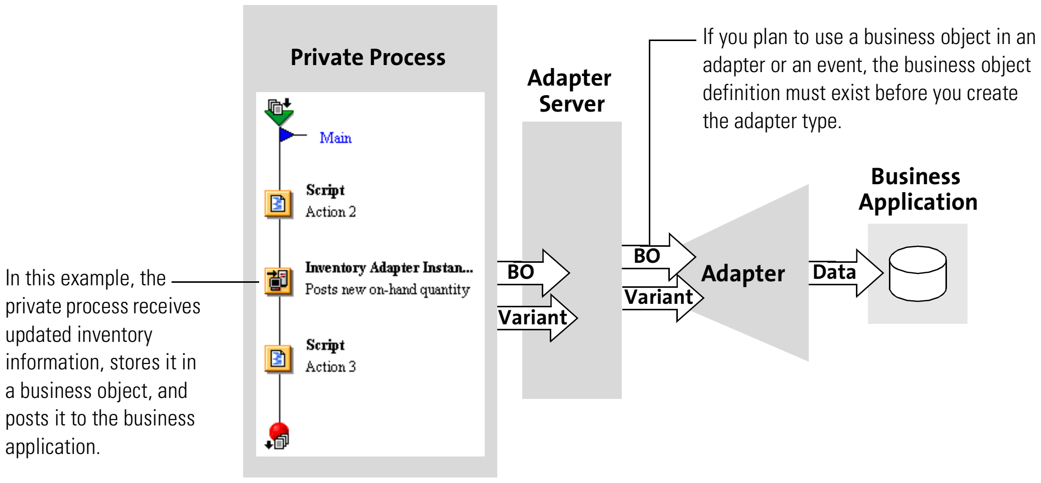
Before you design an adapter, it's important to understand how the different components of an adapter work together.

These components are:

- business objects or variants used to receive information from and pass information to Partner Agreement Manager
- operations the adapter performs
- events the adapter generates
- properties that can be set for each instance of the adapter

About business objects and variants

Partner Agreement Manager manages information in the form of business objects (BOs) or variants. An interchange between Partner Agreement Manager and another application usually has a business object component. However, depending on the type of operations the adapter performs, you might also use variants. Business objects are better for structure information, while variants are better for scalar values.


Private Process
Main
Script
Action 2
Inventory Adapter Instan...
Posts new on-hand quantity
Script
Action 3
Adapter Server
BO
Variant
BO
Variant
Adapter
Data
Business Application
If you plan to use a business object in an adapter or an event, the business object definition must exist before you create the adapter type.
In this example, the private process receives updated inventory information, stores it in a business object, and posts it to the business application.

The business objects must be defined in Partner Agreement Manager before you create an adapter type that uses them. See the *Partner Agreement Manager User's Guide* for more information about defining business objects.

Tip: You can also use a status business object to return status about each operation an adapter performs. Partner Agreement Manager contains a predefined business object—Extricity.3.Operation_Status.1 BO—that you can use to report operation status (and to allow process designers to handle application error or warning conditions in the private process).

If you do not already have a set of result codes defined, you can use these:

Return this Operation_Status.result	If the operation
success	Completed successfully.
failure	Failed. The Operation_Status.reason field must contain the reason the operation failed.
warning	Completed successfully, but there are problems someone might need to know about. The Operation_Status.reason field must contain warning messages.

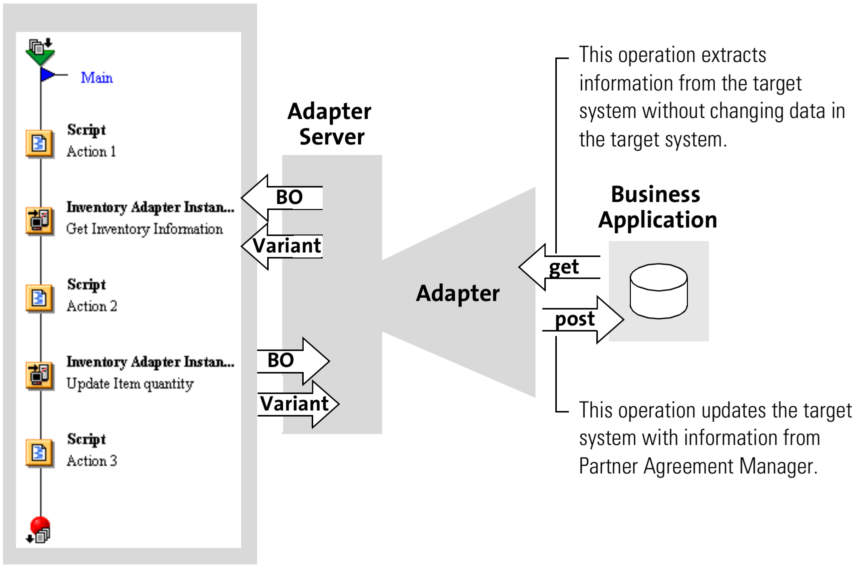
About operations

An operation is a task that the adapter performs for a private process. Partner Agreement Manager adapters can perform two basic types of operations: Get and Post. A third operation, Advanced, can be a combination of gets and posts, or it can be simpler than a Get or Post.

In deciding what type of operation to use, the first step is to determine the objective of the operation:

- Use a Get operation to extract, query, or download information from a business system. A Get operation returns a single business object output, along with an optional status business object.
- Use a Post operation to create, insert, change, update, or delete data in the business system. A Post operation has a single business object input, along with an optional output and status business object.

- Use an Advanced operation to span several Get/Post actions with a single operation. An Advanced operation can have several business object and variant inputs and outputs.


Main
Script
Action 1
Inventory Adapter Instan...
Get Inventory Information
Script
Action 2
Inventory Adapter Instan...
Update Item quantity
Script
Action 3
Adapter Server
BO
Variant
BO
Variant
Adapter
get
post
Business Application
This operation extracts information from the target system without changing data in the target system.
This operation updates the target system with information from Partner Agreement Manager.

Post operations are stateful, and Get operations are usually stateless. For example, a Get operation extracts information from the target system without modifying the target system's data. A database query is an example of a Get operation. In contrast, a Post operation updates the target system. For example, a Post operation can create a new sales order in a purchasing system.

In some cases a Get operation can modify a system's state. For example, a Get operation might file a request on the target system for information that will be fulfilled concurrently. Although this request is logged by the target system, the operation must still be a Get as long as no business information has changed.

Alternately, a query operation can require the services of an intermediary (for example, a middleware product) to contact the target system. From the perspective of the intermediary, a great deal might have happened (it received and fulfilled a request). However, this operation must still be a Get if the end system is in the same state.

If a process is interrupted (for example, the Process Server is restarted while a process is running), the Process Server restarts the process from the top of the current step in the private process. Although it doesn't redo any of the posts, it does redo the gets and stateless Advanced operations. (It calls the reExecute method for all operations.) Remember this when you decide if an operation must be a Get or Post: determine whether it is all right to rerun it.

About immediate and long-running operations

Partner Agreement Manager lets you write long-running operations by calling the requestRetry method of com.extricity.adapter.api.OperationContext interface, which you add to the code yourself.

An immediate operation is one that takes place and completes immediately. A long-running action is one that blocks the private process for a period of time. You can use immediate or long-running operations with Get, Post, and Advanced operations. For example, an immediate Get operation returns immediately if a purchase order is not present in an enterprise system. A long-running Get operation would block until the purchase order appears in the system.

Tip: It's a good idea to use operation names and descriptions that clearly reflect whether the operation will be performed immediately or might block for a time period.

One style of interaction might generally be more applicable for a given business object. For example, you would likely want to get a purchase order immediately, but create a long-running Get for an operation that waits for purchase order approval. You might want to implement both immediate and long-running operations if both are likely to be useful for process designers.

The way in which you combine long-running and immediate gets and posts depends on the tasks that the adapter will perform. For example, if you have stateful interactions that don't complete immediately, you can create an adapter that contains a long-running Post or an immediate Post and a long-running Get. Whether you use a long-running or immediate Post depends on whether everything must be completed in a single transaction.

- Action plan one: one long-running Post that starts the work and waits until it is done.

  Use this plan if everything must be completed as one transaction. Also, if the appropriate context information is difficult to pass back to the private process for a subsequent long-running Get, do all the work as one action.

- Action plan two: immediate Post that starts the work, followed by a long-running Get that waits for the work to be done. This requires the first Post to pass enough context information back to the private process for the second Get to know what operation to check for completion.

  Use this more modular plan to give process designers the opportunity to do other useful work before the task of the original Post completes. In addition, the first Post can be used by itself if the process designer does not care when the operation finishes. The second Get can be used by itself if the process designer wanted to wait for a business object created by an external entity.

About events

Events are occurrences that take place in the business system, such as the arrival of a purchase order, or falling below a predetermined inventory level. Process designers might use events as triggers to initiate Partner Agreement Manager processes. Events in adapters provide a mechanism for the event to pass from the business system to the Process Server. The Process Server can accept event information in the form of business objects or variants.

Although you can achieve the same basic results by creating an event with business object input or by using a Get at the beginning of a process, there are some cases when you need to have a business object input (for example, when an event in a business system can only be captured in a single transaction). Most of the time, however, it's a better idea to start a process with an identifying variant and have the first private action get the business object using that variant.

The advantages to this approach are:

- It makes process testing easier (it's easier to enter a variant manually than a business object).
- You might be able to reuse the Get functionality in another process.
- The private process can handle errors much better than in event generation. So, you want to do as little as possible when generating an event.
- It makes better use of memory. Event inputs are kept in memory for the entire life of the public process. Private process business objects are kept in memory only while the private process is running.

About properties

Properties have values that can be different for separate instances. Instances behave differently depending on these values.

For example, you might have an adapter type that interacts with a database. The adapter type would contain properties that identify the database, the database user, and the user's password. If you wanted to connect to several databases, you can create a different instance of the adapter for each database you connect to. Each instance would use the appropriate database name, user name, and password. The adapter type specifies that database name, user name, and password are properties; it is in the adapter instance that you specify exactly what the database name, user name, and password values are.

When you add properties to an adapter, you can specify that the property is:

- mandatory or optional
- encrypted (for security purposes)
- constrained to a predefined list of values
- a particular data type (boolean, integer, or string)
- set to a default value

CHAPTER
4

Creating an Adapter Type

Read this chapter for information about creating adapter types, importing and exporting adapter types and implementations.

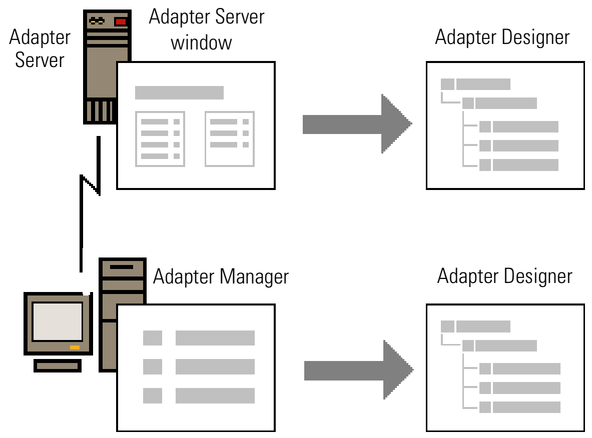
This chapter includes these sections:

About adapter types

An adapter type serves as the blueprint for the implemented adapter. The adapter type defines the operations the adapter performs, the events it conveys from the business system, and the properties it uses. Each type of element has its own set of required information that defines it.

Starting the Adapter Designer

You can start the Adapter Designer from the Adapter Server window or the Adapter Manager:



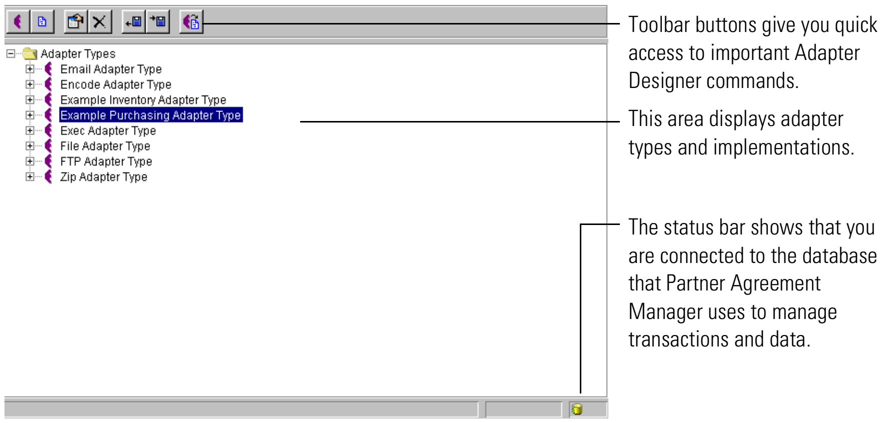
To start the Adapter Designer:

▶ In the Adapter Server window or the Adapter Manager window, choose Adapter Designer from the Tools menu.

The Adapter Designer appears.

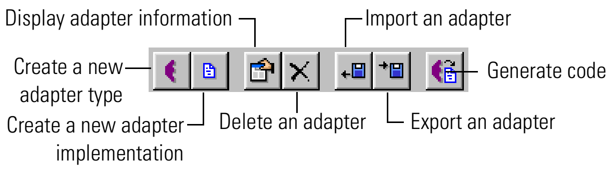
About the Adapter Designer

You use the Adapter Designer to create or modify adapter types, create adapter implementations, or import or export adapter types or implementations. The first time you start it, the Adapter Designer window shows the Example and Utility adapter types. Thereafter, it also displays the adapter types and implementations that you created.


Toolbar buttons give you quick access to important Adapter Designer commands.
This area displays adapter types and implementations.
The status bar shows that you are connected to the database that Partner Agreement Manager uses to manage transactions and data.

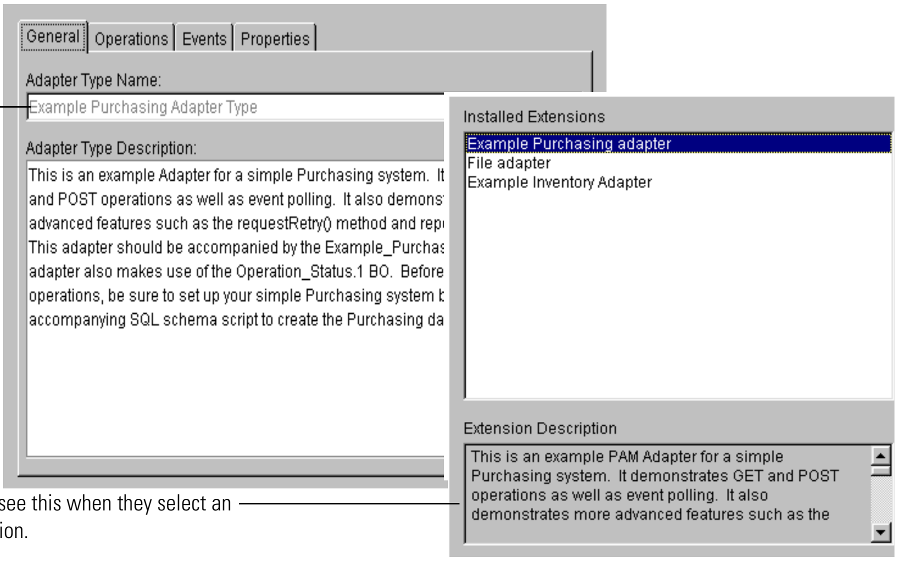
Adapter Types
- Email Adapter Type
- Encode Adapter Type
- Example Inventory Adapter Type
- Example Purchasing Adapter Type
- Exec Adapter Type
- File Adapter Type
- FTP Adapter Type
- Zip Adapter Type

The Command toolbar at the top of the Adapter Designer lets you do the following:


Display adapter information
Import an adapter
Create a new adapter type
Generate code
Create a new adapter implementation
Delete an adapter
Export an adapter

Defining a new adapter type

When you define a new adapter type, you begin by supplying a name and description. Process designers see the description when they select an adapter as part of a private process action. In addition, the description you provide becomes Javadoc documentation for the compiled adapter class.



For this adapter type...

Example Purchasing Adapter Type

This is an example Adapter for a simple Purchasing system. It
and POST operations as well as event polling. It also demons
advanced features such as the requestRetry() method and rep
This adapter should be accompanied by the Example_Purchas
adapter also makes use of the Operation_Status.1 BO. Before
operations, be sure to set up your simple Purchasing system b
accompanying SQL schema script to create the Purchasing da

This is an example PAM Adapter for a simple Purchasing system. It demonstrates GET and POST operations as well as event polling. It also demonstrates more advanced features such as the

...private process designers see this when they select an adapter for an Extension action.

Next, you add the operations that the adapter will perform. For each operation you add, specify a name, a description, inputs, and outputs. See *Adding operations* on page 46 for more information.

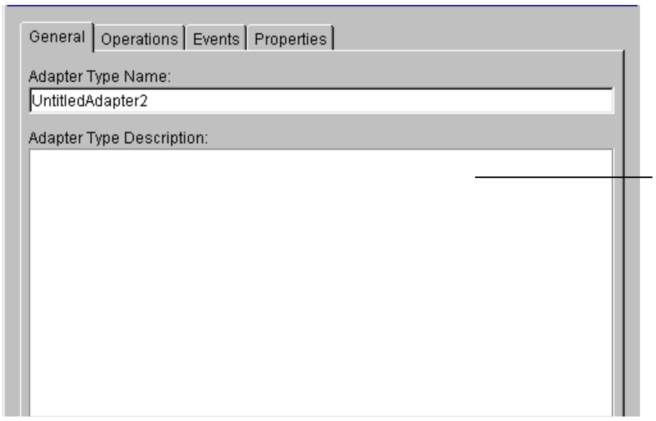
After you add operations, define the events the adapter uses. For each event, you specify a name, a data type, and a time-out value. See *Adding events* on page 51 for more information.

Finally, you define the adapter's properties. For each property, you specify a name and description, define the type, default value, and constraints, and determine whether properties are mandatory or encrypted. See *Adding properties* on page 54 for more information.

To define a new adapter type:

1 In the Adapter Designer, choose Create Adapter Type from the File menu or click the Create Adapter Type button in the Command toolbar.

The Create Adapter Type dialog box appears.



Adapter Type Name: UntitledAdapter2

Adapter Type Description:

The description becomes Javadoc documentation for the adapter implementation.

2 Type a name for the adapter and enter a description.

See *Adding operations* on page 46 to continue defining the adapter type.

About operations

There are three types of operations: Get, Post and Advanced.

A Get operation retrieves information from the end system. Every Get operation has the following characteristics:

- It doesn’t change the end system.
- It has a single output business object.
- It can have unlimited input business objects and/or variants.
- It can have an optional output status business object.

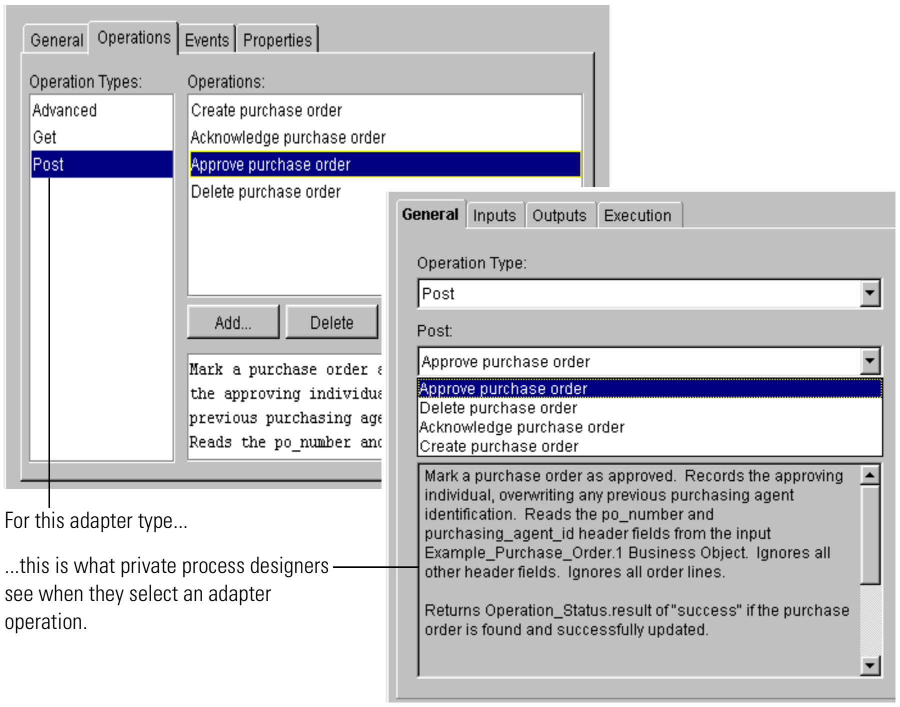
A Post operation puts information into the end system. Every Post operation has the following characteristics:

- It changes the end system.
- It has a single input business object.
- It has an optional output business object.
- It has an optional output status business object.

An Advanced operation is an operation that is a combination of gets and posts or anything else that is not either a Get or a Post.

Adding operations

For each operation you add, you provide an operation name and description. When process designers use this adapter, they select which operations to use based on the operation name and description.



```
Mark a purchase order a
the approving individua
previous purchasing age
Reads the po_number and
```

Operation Type: Post

Post: Approve purchase order

- Approve purchase order
- Delete purchase order
- Acknowledge purchase order
- Create purchase order

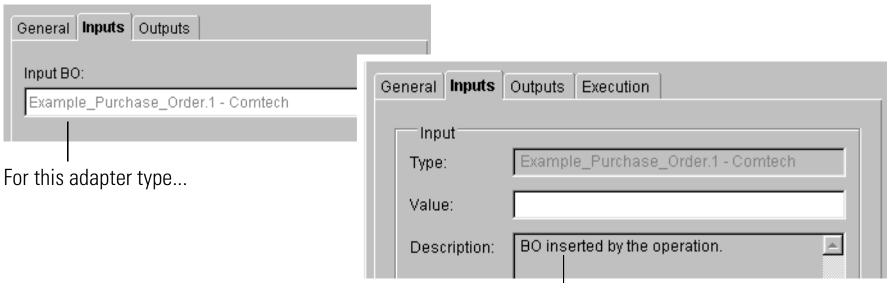
Mark a purchase order as approved. Records the approving individual, overwriting any previous purchasing agent identification. Reads the po_number and purchasing_agent_id header fields from the input Example_Purchase_Order.1 Business Object. Ignores all other header fields. Ignores all order lines.

Returns Operation_Status.result of "success" if the purchase order is found and successfully updated.

For this adapter type...

...this is what private process designers see when they select an adapter operation.

Each adapter operation also has its own set of inputs and outputs that you specify. Depending on the type of operation you add, inputs are either business objects or variants that hold values that the adapter needs to start the operation. Outputs are business objects or variants that receive the results and status for the operation. Private process designers supply context variables that hold the inputs and outputs you specify in the adapter type.


General
Inputs
Outputs
Input BO:
Example_Purchase_Order.1 - Comtech
General
Inputs
Outputs
Execution
Input
Type:
Example_Purchase_Order.1 - Comtech
Value:
Description:
BO inserted by the operation.

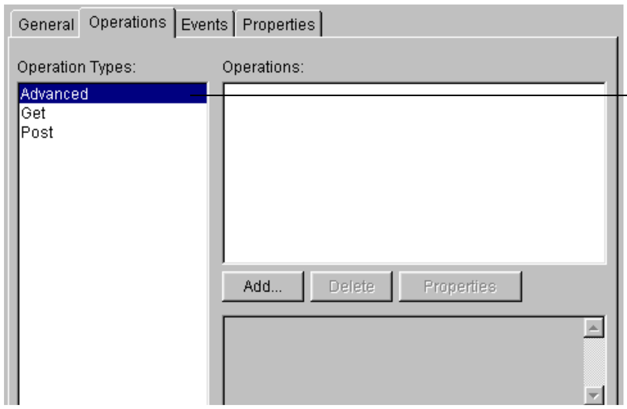
For this adapter type...

...private process designers see this when they specify inputs or outputs.

To add operations:

1 In the Create New Adapter Type dialog box, click the Operations tab.

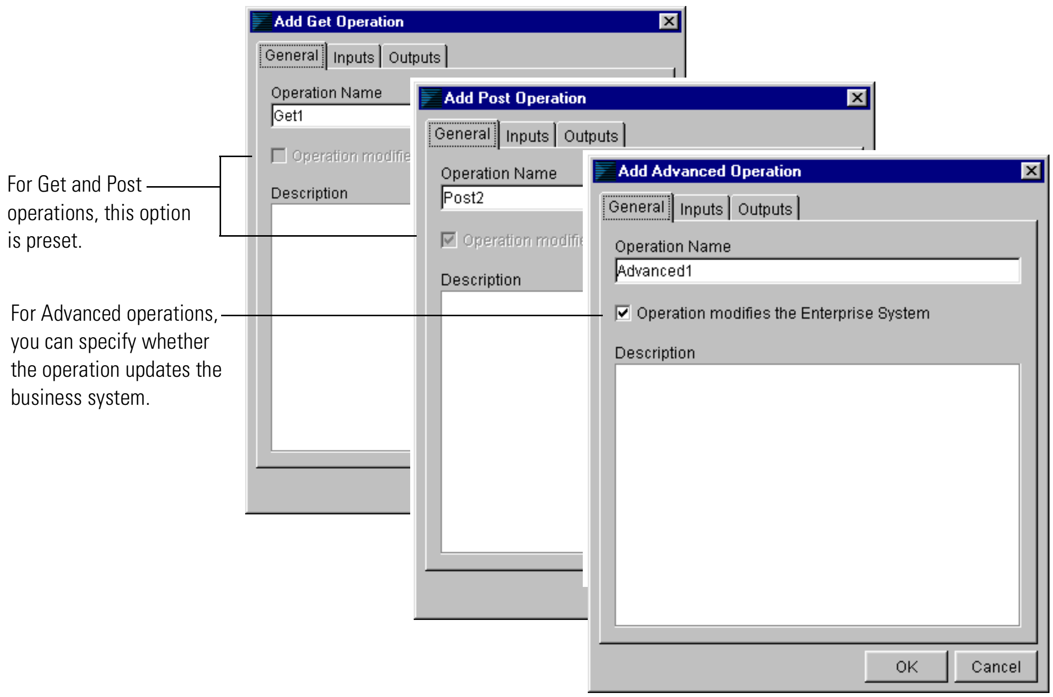
The Operations panel appears. It lists the three types of operations you can add: Advanced, Get, and Post.



Select an operation type.

2 Select an operation type and click Add.

The Add Operation dialog box appears.


For Get and Post operations, this option is preset.
For Advanced operations, you can specify whether the operation updates the business system.
Add Get Operation
General
Inputs
Outputs
Operation Name
Get1
Operation modifie
Description
Add Post Operation
Post2
Add Advanced Operation
Advanced1
Operation modifies the Enterprise System
OK
Cancel

3 Type a name and description for the operation.

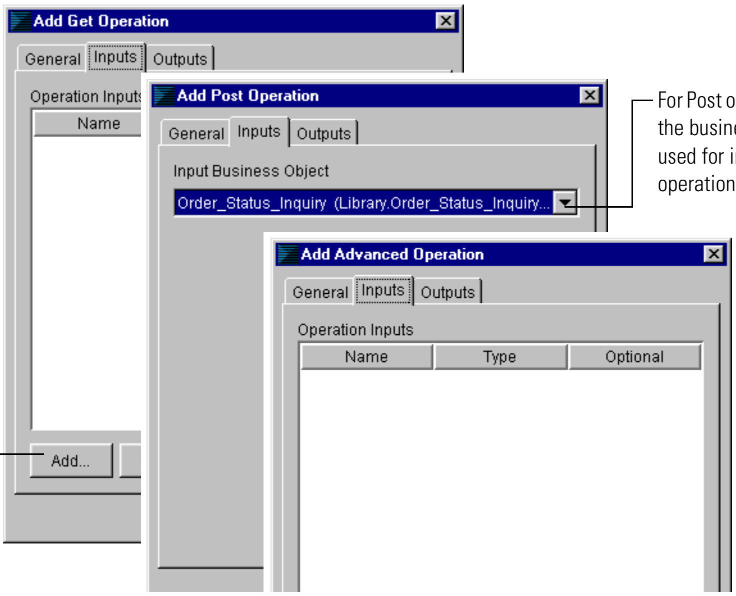
It's a good idea to use operation names that reflect whether the operation will be performed immediately or might block for a period of time. For example, you might want to call an immediate Get operation "Get purchase order," and call a similar long-running Get operation "Wait for purchase order." You must also use the operation description to clarify how long the operation takes so that process designers are aware of these timing issues.

4 For an Advanced operation, specify whether the operation modifies the business system.

When you create implementations of this adapter type, the code generator supplies different re-execution code depending on whether the operation modifies the business system.

5 Click the Inputs tab to specify the input variable or business object for the operation.

The Inputs panel appears. For Post operations, you specify the input business object that contains the information that will be used to update the business system. For Get and Advanced operations, you can specify variables that will hold information for the business system.



Input Business Object: Order_Status_Inquiry (Library.Order_Status_Inquiry...

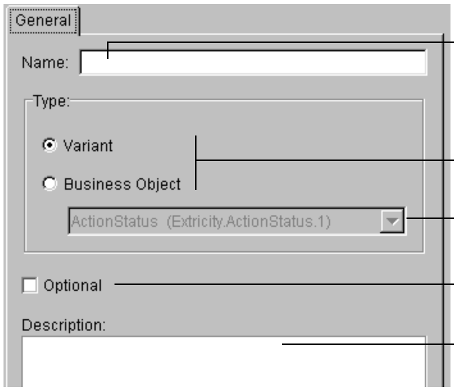
For Post operations, specify the business object to be used for input into the operation.

For Get and Advanced operations, click Add to specify a variable to be used to hold the input for the operation.

6 Specify the input for the operation.

- For Post operations, select a business object from the Input BO list.
- For Get or Advanced operations, click Add.

The Add Input/Output dialog box appears.



ActionStatus (Extricity.ActionStatus.1)

Type a name for the input. Process designers will bind a variable to the input.

Select the type of variable.

If you specify a business object variable, select the business object.

If the input is not required, turn on the Optional setting.

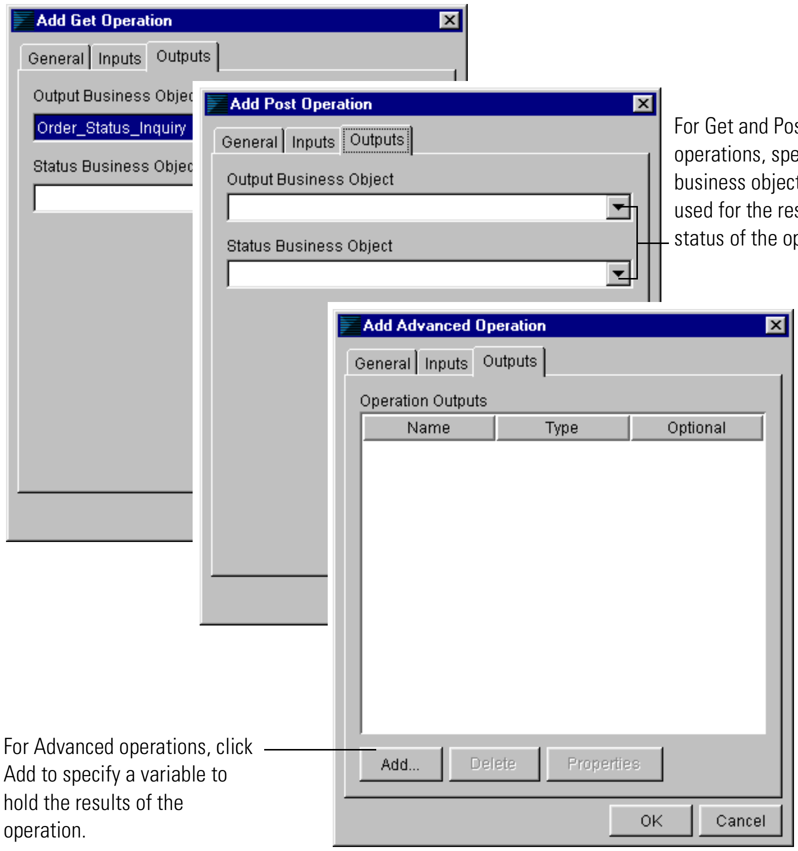
Type a description for the input. The description becomes the Javadoc description of the input or output in the Java implementation.

NOTE: You can add more than one input variable (of either the business object or variant type) to a Get or Advanced operation.

7 Click the Outputs tab to specify the output variable or business object for the operation.

The Outputs panel appears. For Get and Post operations, you can specify one business object that contains the information from the business system, and a second business object that contains operation status. An output business object is required for Get operations, but is optional for posts. Although status business objects are optional for both Get and Post operations, they can be very useful to private process designers.

For Advanced operations, you can specify one or more variables that will hold the results of the operation.



For Get and Post operations, specify the business objects to be used for the results and status of the operation.

For Advanced operations, click Add to specify a variable to hold the results of the operation.

8 Specify the outputs for the operation.

- For Get and Post operations, select a business object from the Output Business Object and Status Business Object lists.
- For Advanced operations, click Add and use the Add Input/Output dialog box to specify one or more variables.

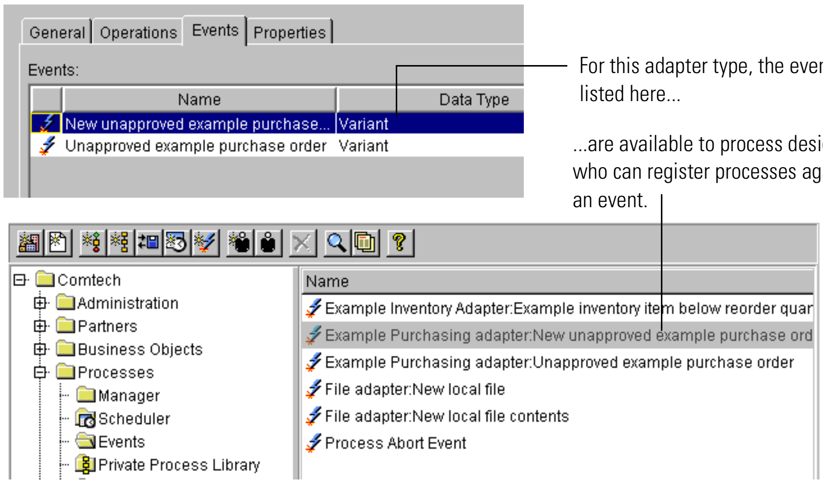
See *Adding events*, next, to continue defining the adapter type.

Adding events

Events represent functional occurrences that take place in the business system, such as the arrival of a purchase order or falling below a predetermined inventory level. The function of events in adapters is to provide a mechanism for the occurrence to pass from the business system to the Process Server. Once an event arrives at the Process Server, process designers can use it as a trigger to start a PAM process. The Process Server can accept event information in the form of business objects or variant variables.

Important: If your event contains a business object, it's a good idea to use a business object that you, and not a partner, own. You don't want your adapter operations to be impacted by your partner's decisions about business object design.

The events that you define in the adapter type appear in the Process Manager window and are available to process designers.



Name	Data Type
New unapproved example purchase...	Variant
Unapproved example purchase order	Variant

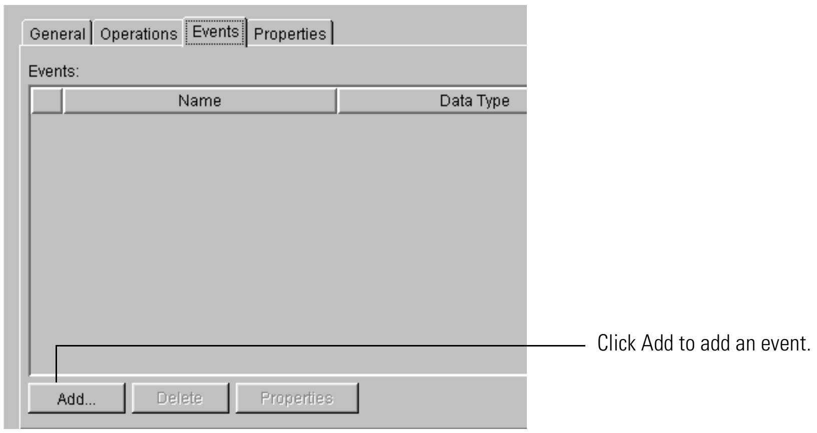
Name
Example Inventory Adapter:Example inventory item below reorder quar
Example Purchasing adapter:New unapproved example purchase ord
Example Purchasing adapter:Unapproved example purchase order
File adapter:New local file
File adapter:New local file contents
Process Abort Event

When you add events, you specify a name for the event, a variable that stores event information, and a time-out that determines how long the event is available to trigger process instances.

To add events:

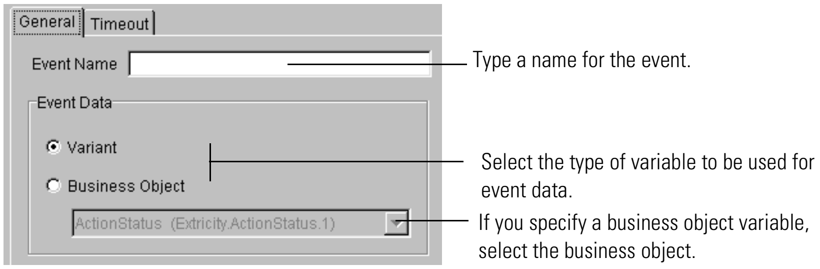
1 In the Create New Adapter Type dialog box, click the Events tab.

The Events panel appears.



2 Click Add to add a new event.

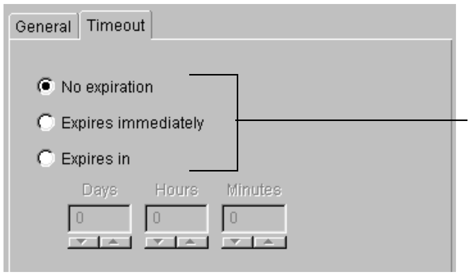
The Add New Event dialog box appears.


General
Timeout
Event Name
Event Data
Variant
Business Object
ActionStatus (Extricity.ActionStatus.1)
Type a name for the event.
Select the type of variable to be used for event data.
If you specify a business object variable, select the business object.

3 Type a name for the event and select the type of variable to be used for the event data. If you select a business object variable, select the business object to be used.

4 Click the Timeout tab to set timing properties for the event.

The Timeout panel appears.



(●) No expiration
( ) Expires immediately
( ) Expires in

Days	Hours	Minutes
0	0	0

Select how long the event remains available.

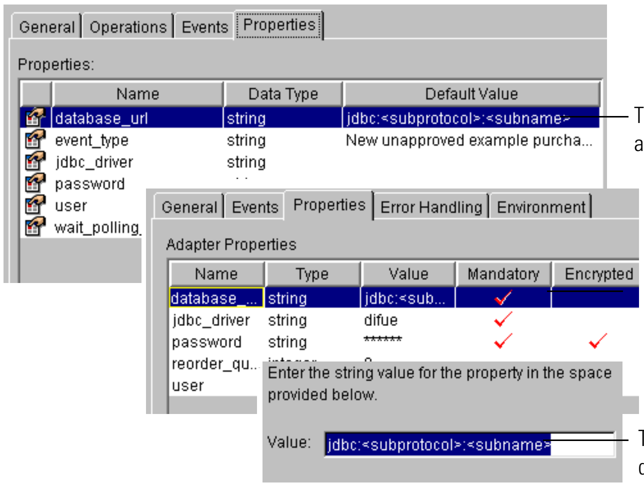
The time-out specifies how long an event remains valid. In other words, the Process Manager has that amount of time to start a public process. Realize that if a public process is suspended, the Process Manager will not start this public process until it is resumed; so if the process is resumed before the event time-out, the Process Manager can start the process.

5 Click OK in the Add Event dialog box, and continue to add events as needed.

See *Adding properties*, next, to continue defining the adapter type.

Adding properties

Properties are values you can use to distinguish between different instances of an adapter. For example, you might have one adapter that can be used to connect to different resources. In the Adapter Manager, you can install the adapter several times, creating several instances of the adapter—once for each resource you connect to. And each time you install the adapter and create an instance, you would specify a different set of properties for the resource you're connecting to.



Properties:

Name	Data Type	Default Value
database_url	string	jdbc:<subprotocol>:<subname>
event_type	string	New unapproved example purcha...
jdbc_driver	string	
password		
user		
wait_polling		

Adapter Properties

Name	Type	Value	Mandatory	Encrypted
database_...	string	jdbc:<sub...	✓	
jdbc_driver	string	difue	✓	
password	string	******	✓	✓
reorder_qu..				
user				

Enter the string value for the property in the space provided below.

Value: jdbc:<subprotocol>:<subname>

The properties for this adapter type...

...determine the properties when you add an instance of this adapter in the Adapter Manager.

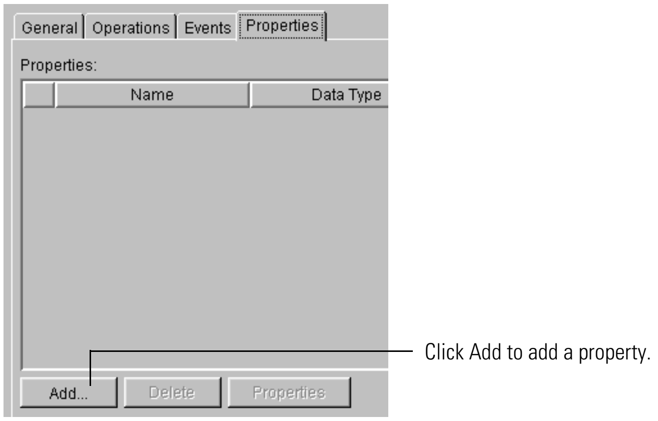
The adapter properties also determine the nature of the values you can enter.

When you add properties to an adapter, you specify a name and description, and whether they are mandatory or optional. You can also specify a data type, provide a default value, constrain a property to a predefined list of values, or designate it as encrypted.

To add properties:

1 In the Create New Adapter Type dialog box, click the Properties tab.

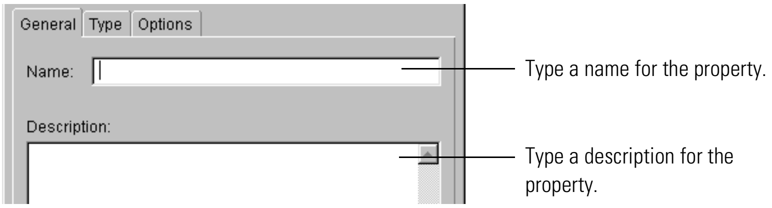
The Properties panel appears.



Click Add to add a property.

2 Click Add to add a new property.

The Property Editor dialog box appears.



Type a name for the property.

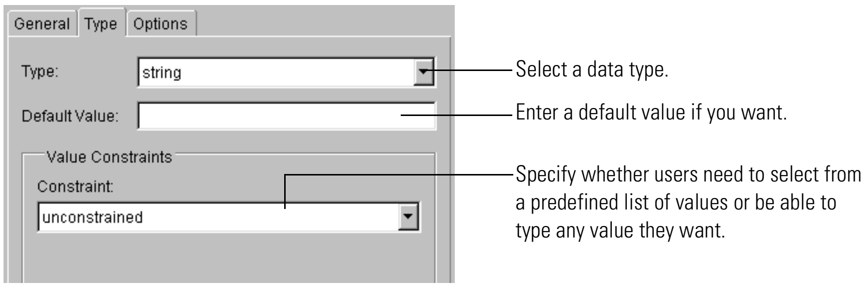
Type a description for the property.

3 Type a name and a description for the property.

The description becomes the Javadoc documentation for the property in the Java implementation.

4 Click the Type tab to specify data type.

The Type panel appears.



Type: string

Default Value:

Constraint: unconstrained

Select a data type.

Enter a default value if you want.

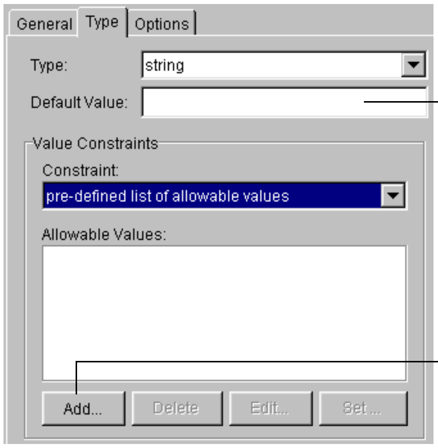
Specify whether users need to select from a predefined list of values or be able to type any value they want.

5 Select a data type (string, integer, or boolean) and enter a default value if you want.

The default value appears whenever users install an instance of this adapter.

6 Specify whether you want users to select from a predefined list of values or be able to type any value they choose.

If you select Pre-defined List Of Allowable Values, the Allowable Values panel appears.



Type: string

Default Value:

Value Constraints

Constraint: pre-defined list of allowable values

Allowable Values:

After you define allowable values, you can select one value and make it the default that appears when users install the adapter.

Click Add to add a value to the list of allowable values.

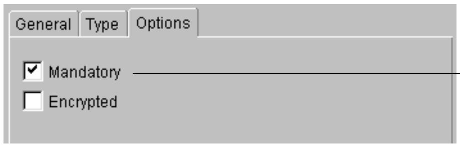
7 Click Add to add a value to the list, type a value in the Add Constraint dialog box, and click OK.

After you define allowable values, you can select one value and click Set As Default to make it the default that appears when users install the adapter.

Tip: Using a constrained list makes it easier for users to enter the correct property values.

8 Click the Options tab to designate the property as mandatory or encrypted.

The Options panel appears. If a property is mandatory, you must implement it in the adapter instance. A mandatory property is required for each adapter instance. An encrypted property, typically a password, is masked when the user enters it.



[✓] Mandatory

[ ] Encrypted

A mandatory property is always required.

TIP: Do not mark a property as mandatory if it doesn't need to be set for proper adapter operation.

9 Click OK to finish defining the adapter type.

EDITING AN ADAPTER TYPE

You can revise an existing adapter type, even after you create implementations and instances for it.

To edit an adapter type:

▶ In the Adapter Designer, double-click an adapter type.

The Edit Adapter Type dialog box appears. The Adapter Type Name is not editable, but all other characteristics are, including operations, events, and properties.

EXPORTING AN ADAPTER TYPE OR IMPLEMENTATION DECLARATION

To make it easier to work in several locations, Partner Agreement Manager lets you export adapters from one computer and import them elsewhere. You might, for example, develop adapters on a workstation for eventual installation on the computer that runs the Adapter Server. Likewise, you can start developing an adapter on a desktop computer, export it, and continue development on any other computer on which the Adapter Designer is installed.

You can export adapter types or implementation declarations. In either case, Partner Agreement Manager creates an XML file.

To export an adapter type or implementation declaration:

1 In the Adapter Designer, select the adapter type or implementation that you want to export.

2 Click the Export icon in the Command toolbar or choose Export from the File menu.

The Export Adapter dialog box appears. The default name is that of the adapter type or implementation declaration.

3 Type a name for the exported adapter and select a location for it. Click OK.

Partner Agreement Manager writes an XML file for the adapter type or implementation declaration you selected.

TIP: Use file names for exported types and implementations that make it easy to distinguish between them when you import them later. For example, you might name an adapter type PurchasingAdapterType and its implementations GeneratedPurchasingAdapterImp and FinalPurchasingImp.

If you are running as a client, the export file is placed on your local computer.

IMPORTING AN ADAPTER TYPE OR IMPLEMENTATION DECLARATION

After you export an adapter developed on a computer other than the one that runs the Adapter Server, Partner Agreement Manager makes it easy to import the adapter to the location you want. You can import adapter types or implementation declarations. However, you must import an adapter type before you can import any of its implementation declarations. In addition, in the Adapter Manager, you can import adapter instances that were exported.

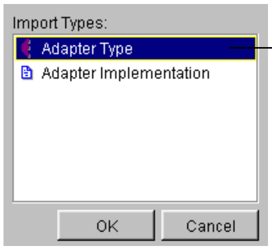
If you are importing adapters to a different Partner Agreement Manager instance (for example, if you're importing an adapter created by a third party), make sure that the business objects used by the adapter are available. If the underlying Process Server is the same, the business objects will already be available. However, events are created on import.

NOTE: You can only import adapters when you are connected to the PAM database (a database icon appears in the lower right corner of the Adapter Designer's status bar). For example, you can't import adapters on your desktop computer when you run the Adapter Designer as a stand-alone application (a disk icon appears instead of the database icon).

To import an adapter:

1 In the Adapter Designer, click the Import icon in the Command toolbar or choose Import from the File menu.

The Select Import Type dialog box appears.



Select the kind of item you want to import. Partner Agreement Manager checks to make sure that the file you select is of the same kind.

2 Select Adapter Type or Adapter Implementation from the Import Types list. Click OK.

The Import dialog box appears.

3 Select the file to be imported. Click Open.

The imported adapter type or implementation appears in the Adapter Designer.

TIP: If you're running as a client, the import file must be on your local computer.

CHAPTER 5

Creating Adapter Implementations

Read this chapter for information about creating adapter implementations in the Adapter Designer.

This chapter includes these sections:

- *About creating adapter implementations* on page 62.
- *Generating code for an adapter implementation* on page 63.
- *Adding Java code* on page 67.
- *Compiling and debugging* on page 67.

For information on adding custom code to adapter implementations, see *Using adapter API methods* on page 103, *Using Business Object API methods* on page 191, and *Adding custom code to adapters* on page 247.

About creating adapter implementations

An adapter implementation is a Java source file that is generated by the Adapter Designer based on the information specified in the adapter type and implementation declaration. It contains Java code with empty methods for the operation and event checking functionality you want to implement and code that loads properties. For example, you create an implementation of an inventory adapter type. The implementation contains empty methods that are placeholders for the code that you'll write to allow it to get and post inventory data.

Creating an adapter implementation is a multi-step process:

Step 1 Generate the code from the adapter type. The Adapter Designer generates a Java file with empty methods.

Step 2 Add Java code for any necessary application logic.

Step 3 Compile the adapter implementation.

Step 4 Create an instance of this adapter.

Step 5 Test and debug the implementation.

The code generated by the Adapter Designer contains placeholders for the operations and events, and definitions for properties you specify. It also contains a method that loads the property values at run time.

A single adapter type can have more than one implementation. For example, you might want to create one implementation for an in-memory test adapter, and another implementation—the real production version—that accesses a database.

Note: Moving from adapter type to implementation is a one-way process. If you revise the adapter type definition after you generate an implementation, you must generate a new implementation to include those changes.

GENERATING CODE FOR AN ADAPTER IMPLEMENTATION

The starting point for creating an adapter implementation is an adapter type. It can be one that you create using the Adapter Designer, or it can be one of the example adapter types that are included in the Partner Agreement Manager Adapter Development Environment (ADE).

NOTE: The example adapters appear in the Adapter Designer. To use one of the example adapters, you need to create your own instance.

You can create as many implementations as you want for a single adapter type. For example, you can develop different implementations of a single adapter type to include varying levels of realism in your testing. Your first adapter implementation can return the same data all the time. A second adapter implementation might preserve data in memory or read and write from flat files.

As you create adapter implementations, you supply class and package names for the implementation that will appear in the generated code. Before you create an implementation, you might want to consider developing a file naming scheme to help you identify the various components for different implementations of the same adapter type.

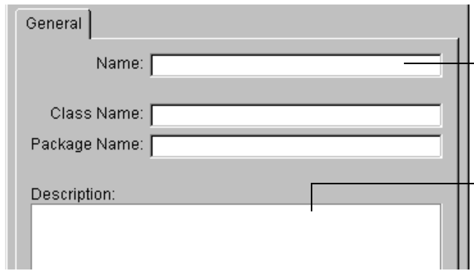
Here's an example:

Use this file name	Under these circumstances
YourAdapterType.xml	This is your exported adapter type information.
YourAdapterJavaImp.xml	This contains the Java implementation declaration, including Java source file name, class name, package, and directory, for your adapter implementation.
YourAdapter.java	This is the file generated by the Adapter Designer code generator.
YourAdapterHelper.java (or YourAdapterCore.java)	This file has all the entry points from the generated adapter implementation into your business logic. (See *Using helpers* on page 248).
SystemLibrary.java	This can be a library for a business system, a technical interface, or an industry-standard protocol.

To generate code for an adapter implementation:

1. Launch the Adapter Designer (if it isn't already running) and select the adapter type for which you want to create the implementation.
2. Choose Create Adapter Implementation from the File menu or click the Create Adapter Implementation button in the Command toolbar.

   The Create Java Adapter Implementation dialog box appears.



Type a class name for the Java implementation of the adapter.

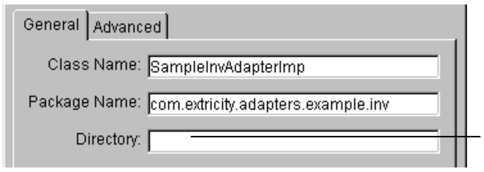
The description becomes the Javadoc documentation for the adapter implementation.

3. Type a name for the adapter implementation, a class name, and a package name. Add a description if you want.

   The class name becomes the name of the Java class file. The package name tells the Process Server where the implementation class file belongs.

   Any description you add becomes internal documentation (Javadoc) for the adapter implementation.
4. Click Generate Code.

   The Generate Java Source Code dialog box appears.



Class Name: SampleInvAdapterImp

Package Name: com.extricity.adapters.example.inv

Directory:

Type the location where you want the implementation class files stored.

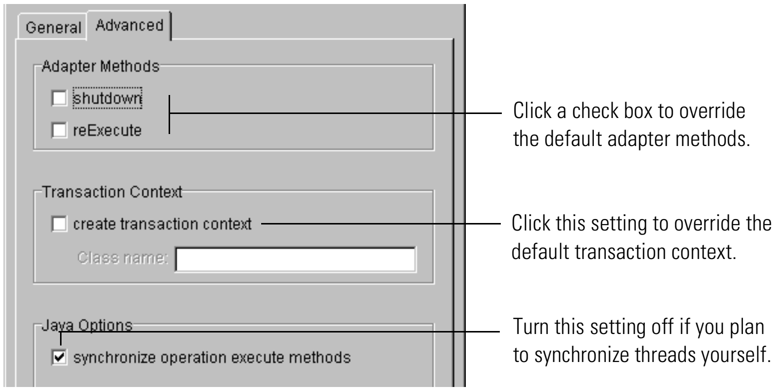
5. Type the location where you want the generated code files to be created.

   The location you enter is relative to the partner directory in your `WebSphere\Partners\Partnernnn`. Remember that if you have a package name, your directory structure must correspond to it for Java to be able to find the files; for example, for `com.mycompany`, use the directory structure `com\mycompany\`.

IMPORTANT: We recommend that you put all adapter files (such as code, XML, and so on) under your partner directory. If you do, during upgrades, Partner Agreement Manager can preserve your files that exist there. In addition, the partner directory is in the Java classpath, so your Java package structure needs to start there.

6 Click the Advanced tab to override default implementation methods.

The Advanced panel appears.



Click a check box to override the default adapter methods.

Click this setting to override the default transaction context.

Turn this setting off if you plan to synchronize threads yourself.

7 Select settings in the Advanced panel as needed.

For any options you select, the Adapter Server generates stubbed code as a placeholder for your custom code, rather than including the default implementation code.

Turn this setting on	To override
shutdown method	The default implementation of the shutdown method in the adapter base class. This method is called when the adapter is stopped. Use this setting to clean up all resources allocated such as database connections. Failure to release resources can result in memory leaks.

Turn this setting on	To override
reExecute method	The default implementation of the reExecute method in the adapter base class. This method is called when execution results are inconclusive due to process or computer failure. Use this setting if you plan to implement special logic for handling reexecution of stateful operations.
Create transaction context	The default behavior of the transaction context. Use this setting when you want to implement your own transaction processing monitor (TP Monitor).

8 If you plan to provide your own multi-threaded synchronization, turn off the Synchronize Operation Execution Methods setting.

IMPORTANT: If you turn this setting off, the generated implementation is not multi-thread safe unless you explicitly handle it yourself. We recommend that you leave this setting turned on in most cases.

9 Click OK.

The Adapter Server generates the code in the directory you specified. You are now ready to add custom code to your implementation, as needed. See *Using adapter API methods* on page 103, *Using Business Object API methods* on page 191, and *Adding custom code to adapters* on page 247.

IMPORTANT: If you are running the Adapter Server on UNIX, you will need to move the generated code to the machine running the Adapter Server. You will also need to update your classpath on the UNIX machine to include the directories that will contain the compiled code. You must do this before you create an adapter instance.

Adding Java code

When you generate code for an adapter, the Adapter Server provides stubbed code for the operations you specify. The generated code contains "To Do" sections that identify the areas of the code that you need to implement.

The next step is to open the generated Java file in a text editor or the integrated development environment (IDE) of your choice and add the required code. The code you add depends on the operations your adapter performs and the interfaces it connects to. Put all initialization, such as attempts to connect to end systems, in the startup method.

Pay special attention to error handling. In the startup method, throw an EndSystemNotAvailableException to indicate that an attempt to connect to a business system failed. This will be caught by the Adapter Server. If error handling was set for the adapter instance, the Adapter Server suspends the adapter and attempts to restart it. For more information on setting error handling and recovery behavior, see the *Partner Agreement Manager Administrator's Guide*.

For more information on writing code for adapters, see *Adding custom code to adapters* on page 247.

Important: If you are running the Adapter Server on UNIX, and you have added code to the adapter, remember to move the updated code files to the machine running the Adapter Server. You will also need to update your classpath on the UNIX machine to include the directories that will contain the compiled code. You must do this before you create an adapter instance.

Compiling and debugging

You need to create an instance of your adapter to test and debug it. See the *Partner Agreement Manager Administrator's Guide*.

CHAPTER

6

Using the JDBC Integration Wizard

Read this chapter for information about using the JDBC Integration Wizard to create an adapter for a supported database.

This chapter includes these sections:

- *Before you start* on page 70.
- *Creating a JDBC Adapter* on page 70.
- *Executing the schemas* on page 80.
- *Running the JDBC adapter* on page 80.

Before you start

You must know what operations, events, and SQL statements you want to include in your adapter before you start using the JDBC Integration Wizard. For each operation you want to include, specify a separate SQL statement. You can build an adapter by entering the SQL statements yourself or specifying stored procedures.

Creating a JDBC Adapter

After you install the JDBC Integration Wizard, make sure that the Process Server and the Adapter Server are running.

Starting the JDBC Integration Wizard

To start the JDBC Integration Wizard:

1 In the Adapter Server, choose Adapter Designer from the Tools menu.

2 In the Adapter Designer, choose Create Adapter Using Wizard from the File menu, and then choose JDBC Integration Wizard from the submenu.

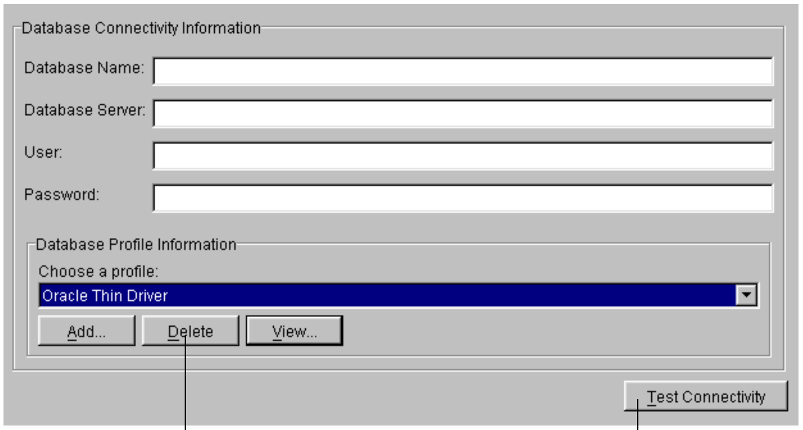
The JDBC Integration Wizard Welcome window appears.

Building a JDBC adapter

To build a JDBC adapter:

1 In the JDBC Integration Wizard Welcome window, click Next to continue.

The Database Connectivity Information window appears.



Click to delete the selected profile.

Check your connection to database server.

2 Enter your connection information.

Type the database name, database server, user name, and password. If you don't know this information, ask your database administrator.

For database profile, you can select an option from the drop-down list (Oracle—Thin Client is the default profile) or create a new profile. To specify a new profile, see step 4.

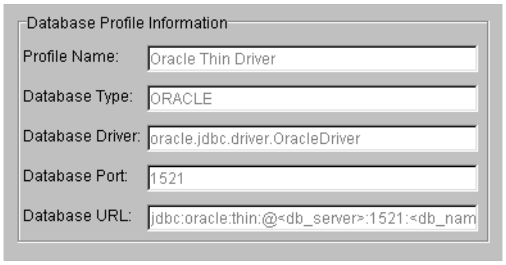
3 Click Test Connectivity to verify that you're connected to your database.

The Information window appears. If you get a connection error, recheck your connection information.

4 You can view or add a profile, if you want.

- Click View to view the selected profile.

  The Database Profile window appears.

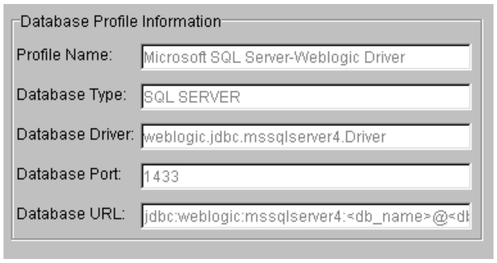


Database Profile Information

Profile Name:	Oracle Thin Driver
Database Type:	ORACLE
Database Driver:	oracle.jdbc.driver.OracleDriver
Database Port:	1521
Database URL:	jdbc:oracle:thin:@<db_server>:1521:<db_nam

- Click Add to create a new profile in the Database Profile window. Enter information for each option.

  The following screen shows example settings for adding a SQL Server profile.



Database Profile Information

Profile Name:	Microsoft SQL Server-Weblogic Driver
Database Type:	SQL SERVER
Database Driver:	weblogic.jdbc.mssqlserver4.Driver
Database Port:	1433
Database URL:	jdbc:weblogic:mssqlserver4:<db_name>@<dt

You might want to create a new profile for different databases or servers. The Database Profile options are described in the following table.

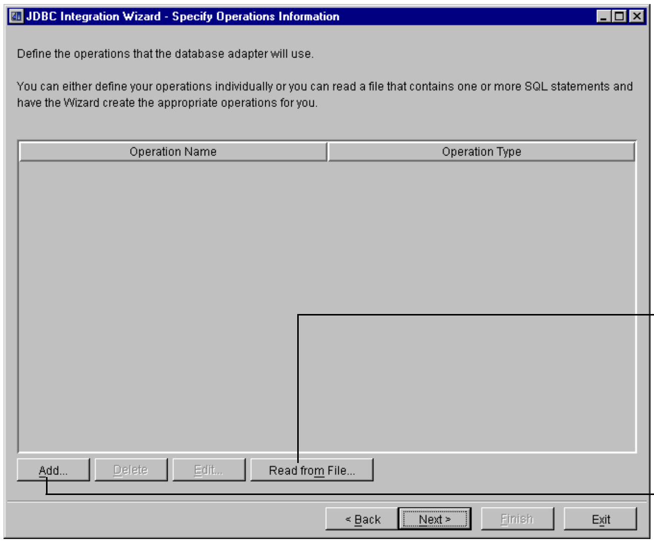
For this option	Supply this information
Profile Name	A unique name for the new profile.
Database Type	**ORACLE** if you're using Oracle. **SQL SERVER** if you're using Microsoft SQL Server.
Database Driver	The database driver (for example, oracle.jdbc.driver.OracleDriver).

For this option	Supply this information
Database Port	The database port number.
Database URL	A generic or specific URL. A generic URL contains placeholders like <*db_name*> and <*db_port*> and can be reused for different databases or servers. The JDBC Integration Wizard populates the URL with specific information for the placeholder tags. For example, if you enter this generic URL for Oracle: jdbc:oracle-thin:@<*db_name*>:<*db_port*>:<*db_server*> or this generic URL for SQL Server: jdbc:weblogic:mssqlserver4:<*db_name*>@<*db_server*>:<*db_port*> The JDBC Integration Wizard provides a specific database name, server, and port. For example: jdbc:oracle-thin:@DELIA:1585:CASSIDY or jdbc:weblogic:mssqlserver4:west@august:1420

NOTE: The Database Profile options are case-sensitive.

5 When you finish, click OK to return to the Database Connectivity Information window, and then click Next.

The Specify Operations Information window appears, where you define operations for the adapter. Although you can define as many operations as you want, you must define at least one operation before you can build the adapter.


JDBC Integration Wizard - Specify Operations Information

Define the operations that the database adapter will use.

You can either define your operations individually or you can read a file that contains one or more SQL statements and have the Wizard create the appropriate operations for you.

Operation Name	Operation Type

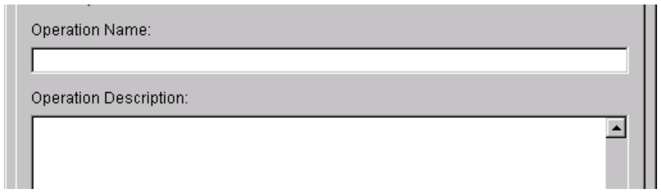
Add...
Delete
Edit...
Read from File...
< Back
Next >
Finish
Exit
Click to add several operations from an imported file of SQL statements.
Click to add an operation.

The following table describes the options in the Specify Operations Information window.

Click	To
Add	Add an operation.
Delete	Delete a selected operation from the list.
Edit	Edit a selected operation in the list.
Read from File	Import a text file containing SQL statements. This file must contain only SQL statements without comments. Separate statements with a semicolon (;) if you're building an adapter for Oracle. Separate statements with the "go" keyword on a separate line if you're building an adapter for SQL Server.

6 Click Add to add an operation.

The Define SQL for Operation window appears with the information in the General panel displayed.



7 In the General panel, type a name for the operation and an optional description. Click the SQL Information tab.

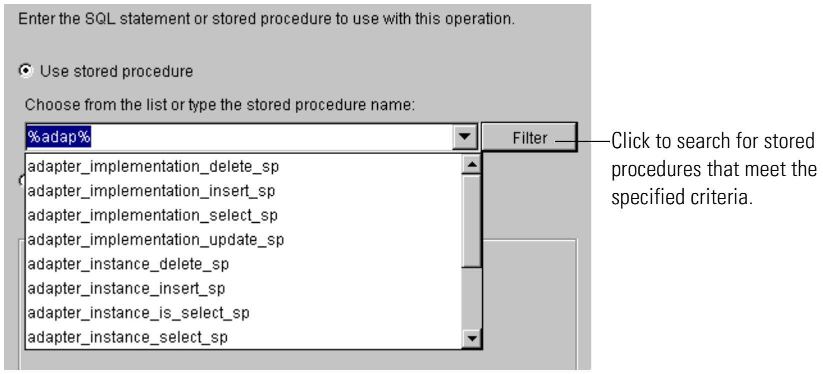
The SQL Information panel appears. You can choose whether to use a stored procedure or type your own SQL statement.

- Click Use Stored Procedure to specify a stored procedure for the operation.

  You can type the name of the stored SQL procedure or filter it using the SQL wildcard character (%). For example, to find tables starting with the string "adapter," type `adapter%` and click Filter. If you want to find tables that include the string "adapter" embedded in their name, type `%adapter%`.

  You can then choose from the filtered options in the dropdown list.

NOTE: Include some criteria along with the wildcard character (%) in your filter. If you use only the %, the filter might take a while to return results, and the list might be too extensive to be useful. Make your search as specific as possible for best results.



Enter the SQL statement or stored procedure to use with this operation.

(●) Use stored procedure

Choose from the list or type the stored procedure name:

%adap%

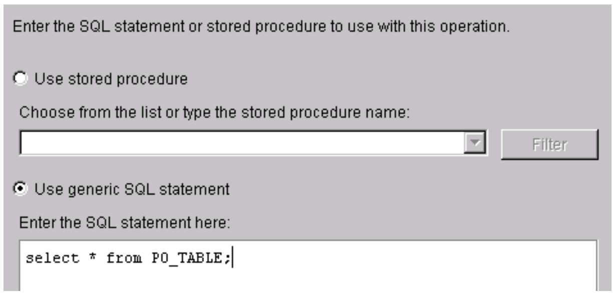
```
adapter_implementation_delete_sp
adapter_implementation_insert_sp
adapter_implementation_select_sp
adapter_implementation_update_sp
adapter_instance_delete_sp
adapter_instance_insert_sp
adapter_instance_is_select_sp
adapter_instance_select_sp
```

Filter
Click to search for stored procedures that meet the specified criteria.

- Click Use Generic SQL Statement to type a SQL statement for the operation. You can enter one Select, Update, Insert, or Delete statement.

  If you want to include a variable in your SQL statement and you're building the adapter for SQL Server, precede the variable with @ (for example, `Select * from employee where empid = @user_defined_id`). If you're building the adapter for Oracle, precede the variable with & (for example, `Select * from employee where empid = &user_defined_id`). End your statement with the "go" keyword if you're using SQL Server or with a semicolon (;) if you're using Oracle.

NOTE: For Oracle databases, table names must be specified in upper case characters.



Enter the SQL statement or stored procedure to use with this operation.

( ) Use stored procedure

Choose from the list or type the stored procedure name:

(●) Use generic SQL statement

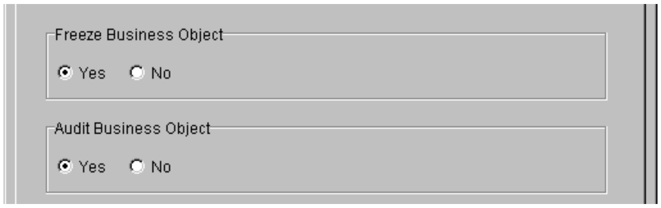
Enter the SQL statement here:

```
select * from PO_TABLE;
```

NOTE: Keep in mind that the JDBC Integration Wizard doesn't validate your SQL statements.

8 Click the Business Object tab to set business object options for the operation.

The Business Object panel appears. You can set options to freeze or audit the business object. The default setting for each option is Yes.

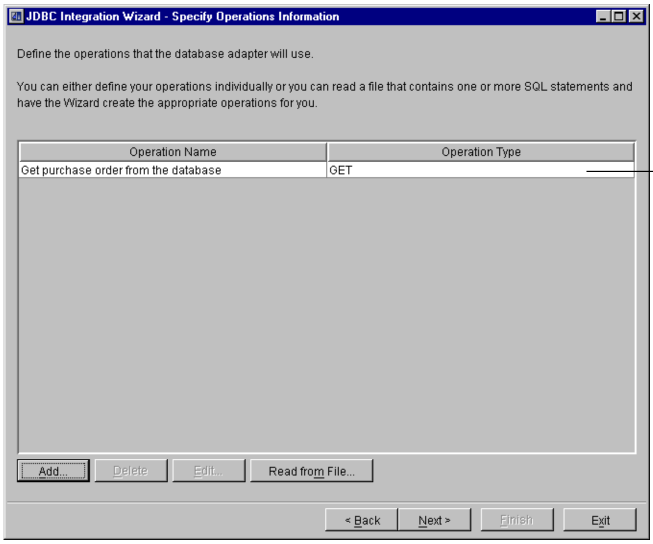

Freeze Business Object

[✓ Yes] No

Audit Business Object

[✓ Yes] No

9 When you finish, click OK to return to the Specify Operations Information window, where you can see the operations. Click Next to continue.


JDBC Integration Wizard - Specify Operations Information

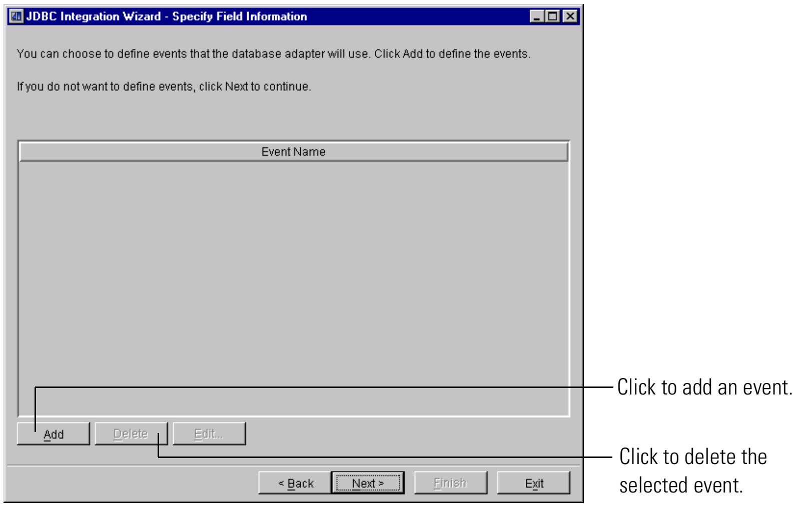
Define the operations that the database adapter will use.

You can either define your operations individually or you can read a file that contains one or more SQL statements and have the Wizard create the appropriate operations for you.

Operation Name	Operation Type
Get purchase order from the database	GET

The operations you add are listed by name and type.

The Specify Event Information window appears, where you can choose to add events for the adapter.


JDBC Integration Wizard - Specify Field Information

You can choose to define events that the database adapter will use. Click Add to define the events.

If you do not want to define events, click Next to continue.

You set up events to be triggered by a specified action (like an Insert, Update, or Delete) on a table in your database. The JDBC Integration Wizard creates the schema for a “shadow table” for the event, with these columns:

- <Pkeys>, the primary key, which might contain one or several columns.
- date_processed, which indicates when a row in the shadow table has been processed by the adapter event.
- processed, which indicates whether a row in the shadow table has been processed by the adapter event.
- trigger_type, explained in the following table.

For this database	This action to the table	Returns this trigger type
SQL Server	Insert or Update	4
	Delete	3
Oracle	Insert	1
	Update	2
	Delete	3

When the event is triggered, the JDBC Integration Wizard uses the table's primary key to populate the shadow table and returns the unprocessed rows of the shadow table.

NOTE: When you add events to your adapter, you must execute the schema on the Oracle or SQL Server database server to create the shadow table and event trigger before you can run the adapter instance. For more information, see *Executing the schemas* on page 80.

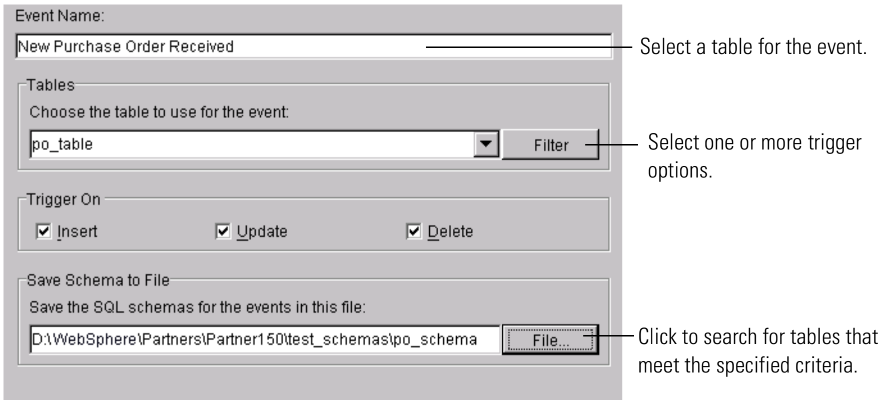
10 Click Add to add an event.

You can specify events, tables, how the events will be triggered, and the file name where the SQL schemas are saved.

The following table describes the options in the Specify Event Information window.

Click	To
Add	Add an event.
Delete	Delete the selected event.
Edit	Edit the selected event.

The Events window appears.


Event Name: New Purchase Order Received

Tables
Choose the table to use for the event: po_table

Trigger On
[✓ Insert] [✓ Update] [✓ Delete]

Save Schema to File
Save the SQL schemas for the events in this file: D:\WebSphere\Partners\Partner150\test_schemas\po_schema

Filter
File...
Select a table for the event.
Select one or more trigger options.
Click to search for tables that meet the specified criteria.

NOTE: For Oracle databases, table names must be specified in upper case characters.

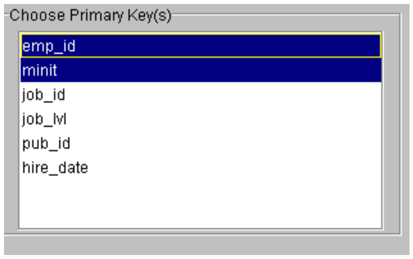
11 Specify Events options, and then click OK to return to the Specify Event Information window. Click Next.

Type any name for the event, select a table to use, and set one or more trigger options (trigger on insert, update, or delete).

Also specify a file name for the SQL schemas that the JDBC Integration Wizard creates. The SQL schema for creating the shadow table and the database trigger to populate the shadow table are saved in the file you specify.

NOTE: If you enable event polling, you (or the database administrator) must execute the schemas after the JDBC Integration Wizard creates the adapter and before the adapter instance is started. For more information, see *Executing the schemas* on page 80.

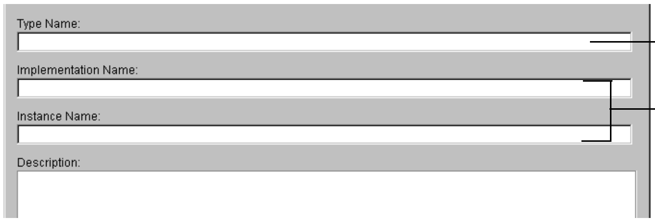
If the specified table doesn't contain a primary key, another window appears when you click OK. You must specify one or more columns for the JDBC Integration Wizard to use as a primary key.



Choose Primary Key(s)

emp_id
minit
job_id
job_lvl
pub_id
hire_date

The Adapter Information window appears.



Type Name:

Implementation Name:

Instance Name:

Description:

Name your adapter.

After you name your adapter and press the Tab key, these fields are filled in for you.

12 Enter a name for the adapter, change the defaults for implementation and instance names as needed, add a description, and then click Finish.

A Progress dialog box appears. The JDBC Integration Wizard creates the adapter type, implementation, and instance for you, as well as any business objects associated with the SQL Select statements you specified.

It’s not necessary to generate code or build your implementation. You can go ahead and add and run the adapter from your private process.

Executing the schemas

If you include events in your JDBC adapter, you must execute schemas in your Oracle or SQL Server database after you generate the adapter and before you start the adapter instance. Only users with the proper permissions (for example, the database administrator) can execute the schemas for events. The schemas must be executed before the adapter instance is started, in this order (see the comments section):

- Execute the CREATE TABLE schema first.
- After the create table schema is executed, execute the CREATE TRIGGER schema.
- Leave the comments in the schema files as is.

Running the JDBC adapter

After you build the adapter, the JDBC Integration Wizard generates and freezes business objects (if you specified the freeze option), and then creates an adapter type, implementation, and instance. Add the adapter instance and business objects to your private process.

The user who designs the private process is responsible for populating the input business object and inspecting the output business object to check the results of the operation.

Creating an event for your adapter adds a new property to the adapter. Be sure to enable event polling in the Adapter Manager.

It’s best to set up test and production instances of your adapter implementation. You can run the private process with the test adapter instance and check the status of the returned business object. For more information on working with adapters, see the *Partner Agreement Manager User’s Guide*.

Troubleshooting tips

Refer to the following list of tips if you encounter problems with the JDBC Integration Wizard.

Building adapters with different drivers

If you use the JDBC Integration Wizard to build more than one adapter using different drivers (for example, if you create your first adapter using the Oracle-Thin Client driver and then make another adapter using a different Oracle driver), you must update the server-side classpath in the Integration Wizard Manager.

Column aliases

You might get unexpected results in your business object fields if you don't specify aliases for columns in the SQL select statement. Non-supported characters (most non-alphanumeric characters) are either stripped out or replaced.

Data types

- Stored procedures can't return REF CURSOR data types.
- Other data types—like BLOB, CLOB, and ARRAY—aren't supported for inserts, updates, deletes, or stored procedures.
- SQL Server doesn't support these data types as part of trigger elements: ntext, text, and image. Any columns with these data types won't appear in the list of choices for primary key.
- JDBC expects these formats for the following data types:
  - Timestamp: `yyyy-mm-dd hh:mm:ss:ffffff...`
  - Date: `yyyy-mm-dd`
  - Time: `hh:mm:ss`

Exporting adapters

If you decide to export and re-import an adapter you created, you will get a password error.

To reset the password:

1. In the Adapter Manager, select the adapter instance to open the Add Adapter Instance window, and then click the Properties tab.
2. Enter a value for the password property, click Set, and then click OK to certify the adapter instance as valid.

CHAPTER

7

USING UTILITY ADAPTERS

Read this chapter for descriptions of the Partner Agreement Manager utility adapters and the business objects they use.

This chapter includes these sections:

About utility adapters

Utility adapters are a group of pre-built adapters designed to perform standard file operations such as reading, writing, and renaming files, file transfers, file encoding or decoding, and file compression. Utility adapters also perform standard e-mail operations and execute commands.

Partner Agreement Manager provides the following utility adapter types:

This utility adapter	Does this
Email adapter	Performs standard e-mail operations such as receiving messages via a POP3 e-mail server and sending e-mail via SMTP.
Encode adapter	Performs standard encoding and decoding operations such as Base64 encoding or decoding, Uuencoding or Uudecoding, and mailing files as MIME attachments.
Exec adapter	Executes the specified command line.
File adapter	Performs standard file manipulation operations such as reading, writing, renaming, moving, and deleting files. It also obtains file attributes, compares two files, and waits for a file to appear in a specific location.
FTP adapter	Performs standard file transfer operations such as downloading a file to a business object or file system or uploading a file from a business object or file system. It also renames or deletes a remote file.
Zip adapter	Deflates or inflates files.

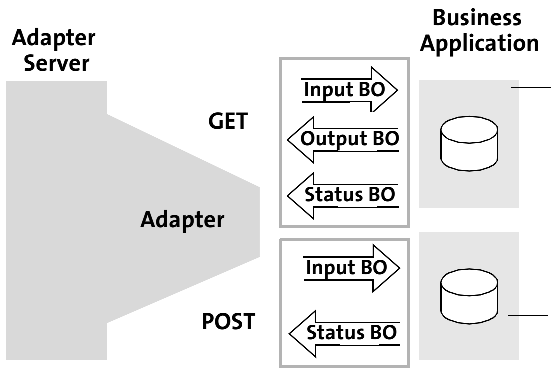
The utility adapters are designed to be used as is, which means that all you need to do is add adapter instances. You cannot change implementations of utility adapters. The types and implementations appear in the Adapter Designer.

For more about creating adapter instances, see *Using the JDBC Integration Wizard* on page 69.

Utility adapter reference

The definitions of the utility adapters contain all the standard elements: properties that can be set for specific instances, operations that can be performed, and events that can be produced.

Depending on its purpose, each utility adapter contains a combination of Get, Post, and Advanced operations. For each operation, the adapters also have business objects specified for inputs, outputs, and status.



In a Get operation, an input business object specifies which information to get and an output business object receives the data. A Get operation can write to status business objects.

In a Post operation, an input business object supplies the information to be used for updating. A Post operation can also write to a status business object.

The sections that follow describe each of the utility adapters, the properties that can be set, the operations it can perform, and the events it can generate.

Email Adapter type

Class name	com.extricity.adapters.utility.email.EmailAdapter
Description	Performs standard e-mail functions such as receiving, sending, and deleting e-mail messages. This adapter supports the Post Office Protocol 3 (POP3 protocol) for receiving messages and the Simple Mail Transfer Protocol (SMTP) for sending messages. It also obtains the number of e-mail messages on the server. Note that POP3 does not support communication through firewalls. This adapter lets you do e-mail notifications that are more flexible than those provided by a notification node in a process. For example, this adapter allows a process to send a notification when the address of the recipient is data-driven. The contents of the Mail_Contents BO can be filled in on-the-fly for maximum flexibility. The events defined in this adapter are used by the checkForEvents method, which moves new mail files to the directory specified by the `processed_directory` property.
Properties	recr_protocol (**required**)—The protocol used to retrieve mail. The only supported value is POP3. sender_protocol (**required**)—The protocol used to send mail. The only supported value is SMTP. recr_host (**required**)—The server name used to retrieve incoming e-mail messages. recr_port (**required**)—The port used to connect to fetch incoming e-mails (default: 110, the default for POP3). sender_host (**required**)—The server host name of the SMTP server used for sending e-mail. sender_port (**required**)—The port number used by the SMTP server (default:25). processed_directory (**required**)—the directory used to save retrieved e-mail. You must have permission to create new files in this directory. user (optional)—The user name for login. If no user name is given, the default login is used. password (optional)—The password for login. If no password is given, the default password is used. event_type (required)—The type of event sent to the Adapter Server. The default is “New mail.” debug_message—The verbosity level for error messages: none, terse or verbose. The default is none.

Operations

POST: Send message

Description: Send a message or send a message with one or more attachments.

Input: Mail_Contents BO

Output: <none>

Status: Operation_Status BO

ADVANCED: Get number of messages

Description: Get the number of messages from the e-mail server for a specified user.

Input: <none>

Output: Number of messages (data type: variant)

Status: Operation_Status BO

ADVANCED: Read message

Description: Reads the first message in the mailbox, saves the e-mail (and attachments) on the local disk and deletes the e-mail from the e-mail server. Attachments are saved in the same directory as the e-mail messages. If two attachments have the same name, the second one overwrites the first.

Input: <none>

Output: Mail_Contents BO

Status: Operation_Status BO

Events

The Email adapter periodically looks for new mail on the server indicated by the recr_host property. If it finds new mail, the adapter moves it from the server to the directory indicated by the processed_directory property. Then the adapter generates a New mail event for each piece of new mail. The Email adapter deletes the new mail from the server once the new mail is saved in the processed_directory.

EVENT: New mail

Description: When the Email adapter finds new mail on the e-mail server, it saves the new mail to the processed_directory and generates a New mail event. The New mail event contains a File_Location BO for each new e-mail message indicating where the new mail is saved.

File_Location BO

EVENT: New mail contents

Description: When the Email adapter retrieves new mail from the server, it generates a Mail_Contents BO for each new e-mail received, containing the contents of the message.

Mail_Contents BO

Encode Adapter type

Class name	com.extricity.adapters.utility.encode.EncodeAdapter
Description	Performs standard encoding and decoding functions such as Base64 encoding or decoding, Uuencoding or Uudecoding, and mailing files as MIME attachments.
Properties	smtp_host (only for “Mail file as MIME attachment”) smtp_from (only for “Mail file as MIME attachment”) uu_permission (optional, only for uuencode) — the UNIX file mode to write on the header (default: 600)
Operations	**POST: Base64 encode file** Description: Encodes an input file in the base64 format. Input: Encode_Operation BO Output: <none> Status: Operation_Status BO
	POST: Base64 decode file Description: Decodes the specified input file in base64 format. Input: Encode_Operation BO Output: <none> Status: Operation_Status BO
	GET: Read, deflate, and base64 encode file Description: Reads an input file, deflates it, and encodes it in the base64 format. Input: File_Location BO Output: File_Contents BO Status: Operation_Status BO
	POST: Base64 decode, inflate, and write file Description: Decodes the string specified in the “contents” part of the input BO in base64 format, inflates it, and writes it out to the specified file. Input: File_Contents BO Output: <none> Status: Operation_Status BO
	POST: Uuencode file Description: Encodes the named input file. If you specify a substitute file name for the uuencoded header, the decoded file uses the substitute name. The default is the input file name. Input: Uuencode_Operation BO Output: <none> Status: Operation_Status BO

POST: Uudecode file

Description: Decodes the named input file and returns the name and location of the decoded file.

Input: Uudecode_Operation BO

Output: File_Location BO

Status: Operation_Status BO

POST: Mail file as MIME attachment

Description: Sends the named input file as an e-mail attachment to the recipient with the specified subject line using the Simple Mail Transfer Protocol (SMTP).

Input: Mail_Attachment BO

Output: <none>

Status: Operation_Status BO

Exec Adapter type

Class name	com.extricity.adapters.utility.exec.ExecAdapter
Description	Executes the specified command line.
Properties	<none>
Operations	**POST: Exec** Description: Supports the execution of external processes. Input: Exec_Operation BO Output: <none> Status: Operation_Status BO
	POST: Blocking Exec Description: Executes an external process, blocks until the process completes, returns stdout and stderr (if it exists). Input: Exec_Operation BO Output: Exec_Output BO Status: Operation_Status BO

File Adapter type

Class name	com.extricity.adapters.utility.file.FileAdapter
Description	Performs standard file manipulation tasks such as reading, writing, renaming, moving, and deleting files. It also obtains file attributes, compares two files, and waits for a file to appear in a specific location. The events defined in this adapter are used by the checkForEvents method, which moves every file/directory in the directory specified by the listen_directory property to the directory specified by the `processed_directory` property.
Properties	listen_directory (for events) processed_directory (for events) callback_polling_period_in_seconds (for the Wait for File operation) event_type (for events)

Operations

GET: Read file

Description: Reads the file specified by the input BO, and stores its contents in the output BO as a string. If the directory is null or empty, Partner Agreement Manager uses the current partner directory.

Input: File_Location BO

Output: File_Contents BO

Status: Operation_Status BO

POST: Write file

Description: Writes the contents of the input BO into the location specified by the input BO. If the file does not exist, Partner Agreement Manager creates the file. If the file already exists, Partner Agreement Manager overwrites the file. If the name of the file is not specified, Partner Agreement Manager will not write the file. If a directory is not specified, but the file name is, Partner Agreement Manager creates the file in the current partner directory.

Input: File_Contents BO

Output: <none>

Status: Operation_Status BO

POST: Delete file

Description: Deletes the file specified in the input BO. If the directory in the input BO is null or empty, Partner Agreement Manager uses the current partner directory.

Input: File_Location BO

Output: <none>

Status: Operation_Status BO

GET: Get file attributes

Description: Gets the specified file's attributes: directory, file name, file size in bytes, last modification date, readable or not, and writable or not. If the directory in the input BO is null or empty, Partner Agreement Manager uses the current partner directory.

Input: File_Location BO

Output: File_Attributes BO

Status: Operation_Status BO

ADVANCED: Are files different?

Description: Compares two files. If file1 and file2 are different, Partner Agreement Manager sets a difference_flag to "true." If the files are the same, Partner Agreement Manager sets a difference_flag to "false."

Input: file_name_1, file_name_2

Output: difference_flag (true or false)

GET: Wait for file (implemented with requestReply)

Description: If the file exists, this operation is the same as a Get: Read file. If the file does not exist, Partner Agreement Manager waits for the number of seconds specified in the `callback_polling_period_in_seconds` property, and then checks for the file again. If the file exists now, Partner Agreement Manager reads the file. Otherwise, Partner Agreement Manager waits for the specified number of seconds and checks the file's existence again. The property `callback_polling_period_in_seconds` must not be shorter than the Adapter Server's `retry_check_interval`.

Input: File_Location BO

Output: File_Contents BO

Status: Operation_Status BO

POST: Rename file

Description: Renames a file to a different directory, to a different file name, or to both. The old file name will no longer make sense after this operation succeeds. If the `from_directory/to_directory` are null or empty, Partner Agreement Manager uses current partner directory. Both to and from file names must exist.

Input: Rename_File_Operation BO

Output: <none>

Status: Operation_Status BO

POST: Move file

Description: Moves a file from one location to another. It can be to a different directory, or to a different file name, or both. The old file location will no longer make sense after this operation succeeds. If the `from_directory/to_directory` are null or empty, Partner Agreement Manager uses the current partner directory. Both `to` and `from` file names must exist.

Input: Move_File_Operation BO

Output: <none>

Status: Operation_Status BO

POST: Copy file

Description: Copies the existing file to a new file. The new file can be in a different directory, can have a different file name, or both. The original file still exists after this operation succeeds. If the `from_directory/to_directory` are null or empty, Partner Agreement Manager uses the current partner directory. Both `to` and `from` file names must exist.

Input: Copy_File_Operation BO

Output: <none>

Status: Operation_Status BO

POST: Append file

Description: Appends the contents in the input BO to the end of the file specified in the input BO. The file specified cannot be read-only.

Input: File_Contents BO

Output: <none>

Status: Operation_Status BO

Events

The File adapter looks for new files on the listen_directory property. If it finds a file, that file is moved to the processed_directory property. Then, depending on the setting of the event_type property, either the New local file event or the New Local File Contents event is generated.

EVENT: New local file

Description: When the event_type property is set to New local file, checkForEvents creates a File_Location BO for each file/directory's new location.

File_Location BO

EVENT: New local file contents

Description: When the event_type property is set to New local file contents, checkForEvents creates a File_Contents BO for each file/directory's new location and contents.

File_Contents BO

FTP Adapter type

Class name	com.extricity.adapters.utility.ftp.FTPAdapter
Description	Performs standard file transfer operations such as downloading a file to a business object or file system or uploading a file from a business object or file system. It also renames or deletes remote files. This FTP adapter also supports FTP passive mode. Passive mode is a state in which the server “listens” on a port that is not its default data port, waiting for a client connection.
Properties	server_host—the FTP server user_name—a user that can log into FTP server password listen_directory (for events) processed_directory (for events) local_directory (for events)—directory on the Adapter Server where downloaded files are placed reconnect_flag—indicates whether the adapter must connect or disconnect for each operation transfer_mode—indicates the mode for FTP transfer. The allowed values are ASCII and BINARY. (Default is BINARY) event_type (for events)—indicates which event to generate
Operations	**GET: Download file to BO** Description: Downloads the contents of the specified remote file into the process context as a part of a BO. Input: File_Location BO Output: File_Contents BO Status: Operation_Status BO
	POST: Upload file from BO Description: Uploads a string specified by the contents field of the input BO to the specified remote file defined by the directory and file_name fields of the input BO. Input: File_Contents BO Output: <none> Status: Operation_Status BO
	GET: Download file to file system Description: Downloads the specified remote file to the local file system. This is analogous to the FTP command Get. Input: FTP_Operation BO Output: File_Location BO Status: Operation_Status BO

GET: Download files to file system

Description: Downloads the specified remote file(s) to the local file system. This is analogous to the FTP command mget.

Input: FTP_Operation BO

Output: File_Location_List BO

Status: Operation_Status BO

POST: Upload file from file system

Description: Uploads a local file to a remote location. This is analogous to the FTP command put.

Input: FTP_Operation BO

Output: <none>

Status: Operation_Status BO

POST: Upload files from file system

Description: Uploads one or more local files to a remote location. This is analogous to the FTP command mput.

Input: FTP_Operation BO

Output: <none>

Status: Operation_Status BO

POST: Delete remote file

Description: Deletes the specified remote file. This is analogous to the FTP command delete.

Input: File_Location BO

Output: <none>

Status: Operation_Status BO

POST: Delete remote files

Description: Deletes the specified remote files. This is analogous to the FTP command mdelete.

Input: File_Location BO

Output: <none>

Status: Operation_Status BO

POST: Rename remote file

Description: Renames the specified remote file.

Input: Rename_File_Operation BO

Output: <none>

Status: Operation_Status BO

Events

EVENT: New FTP file

Moves the new file from the listen_directory to the processed_directory and returns its location on the remote computer.

File_Location BO

	EVENT: New FTP file contents Downloads the remote file contents into a business object and returns the business object. File_Contents BO
	Event: New downloaded FTP file Downloads the remote file to the local file system and returns the location of the file on the local file system. File_Location BO

Zip Adapter type

Class name	com.extricity.adapters.utility.zip.ZipAdapter
Description	Deflates or inflates files.
Properties	<none>
Operations	**POST: Deflate file** Description: Deflates (compresses) the named input file and writes it to the named output file. Input: Zip_Operation BO Output: <none> Status: Operation_Status BO
	POST: Inflate file Description: Inflates (decompresses) the named input file and writes it to the named output file. Input: Zip_Operation BO Output: <none> Status: Operation_Status BO

Utility business object (BO) reference

These are the business objects used by the utility adapters to transport information between Partner Agreement Manager and the target systems.

Copy_File_Operation BO

Description	Used by the File adapter to specify the existing name and location of a file, as well as the new name and location.
Fields	from_directory (optional) from_file_name (**required**) to_directory (optional) to_file_name (optional)

Encode_Operation BO

Description	Used by the Encode adapter to specify the name and location of a file to be encoded or decoded, as well as the location to store the results.
Fields	input_directory (optional) input_file_name (**required**) output_directory (optional) output_file_name (**required**)

Exec_Operation BO

Description	Used by the Exec adapter to specify the command to execute.
Fields	command_line (**required**)

Exec_Output BO

Description	Used by the Exec adapter to report the results of an executed command.
Fields	stdout (optional) stderr (optional)

File_Attributes BO

Description	Used by the File adapter to report the name, location, size, and other attributes of a file.
Fields	directory (optional) file_name (**required**) file_size (**required**) last_modified_date read_flag (true or false) write_flag (true or false)

File_Contents BO

Description	Used by the File, FTP, and Encode, adapters to report the name, location, and contents of a file.
Fields	directory (optional) file_name (**required**) file_contents (**required**)

File_Location BO

Description	Used by the File, FTP, and Encode, adapters to specify the name and location of a file.
Fields	directory (optional) file_name (**required**)

File_Location_List BO

Description	Used by the FTP adapter to specify the name and location of a group of files.
Fields	directory (optional) file_name (**required**) (repeatable)

FTP_Operation BO

Description	Used by the FTP adapter to specify the name and location of a file, as well as its destination name and location.
Fields	local_directory (optional) local_file_name (**required**) remote_directory (optional) remote_file_name (**required**)

Mail_Attachment BO

Description	Used by the Encode adapter to specify the file to be attached to an e-mail message, as well as the recipient and subject of the e-mail.
Fields	smtp_to (**required**) smtp_subject (**required**) file_name (**required**)

Mail_Contents BO

Description	Used by the Email adapter to specify the contents and some header information of an e-mail message.
Fields	from (optional. The default is the user name that logs into the e-mail server.) Who the mail is from. to (**required**, repeatable) The destination addresses the mail is sent to. cc (optional, repeatable) The carbon copy addresses. subject (optional) The subject of the e-mail. contents (optional) The main message body of the e-mail. attached_file_name (optional, repeatable) The file names for each of the attachments. This file name is specified by the full path.

MOVE_FILE_OPERATION BO

Description	Used by the File adapter to specify the existing name and location of a file, as well as the new name and location.
Fields	from_directory (optional) from_file_name (**required**) to_directory (optional) to_file_name (optional)

OPERATION_STATUS BO

Description	Used by each of the adapters to return a status message.
Fields	result (**required**, "success" or "failure") reason (optional)

RENAME_FILE_OPERATION BO

Description	Used by the File and FTP adapters to specify the existing name and location of a file, as well as the new name and location.
Fields	from_directory (optional) from_file_name (**required**) to_directory (optional) to_file_name (**required**)

UUENCODE_OPERATION BO

Description	Used by the Encode adapter to specify the name and location of a file to be encoded, as well as the location to store the results.
Fields	input_directory (optional) input_file_name (**required**) substitute_header_file_name (optional) output_directory (optional) output_file_name (**required**)

Uudecode_Operation BO

Description	Used by the Encode adapter to specify the name and location of a file to be decoded, as well as the location to store the results.
Fields	input_directory (optional) input_file_name (**required**) output_directory (**required**)

Zip_Operation BO

Description	Used by the Zip adapter to specify the name and location of files to be zipped or unzipped, as well as the location to store the results.
Fields	input_directory (optional) input_file_name (**required**) output_directory (optional) output_file_name (**required**)

CHAPTER 8

Using adapter API methods

Read this chapter for reference information on the Adapter API methods that you can use in adapter code. If you are a Java programmer, this chapter gives you the information you need to expand on the Java implementation created by the Adapter Designer.

For information about the Business Object API, see *About the Business Object API* on page 192.

This chapter includes these sections:

About the adapter API methods

Adapters interact with the Process Server and the Adapter Server through the methods in the Adapter API. These are in the com.extricity.adapter.api package. Methods for manipulating business objects (BOs) are in com.extricity.document.api. You can use these packages in your adapter source code. The Adapter Designer adds some methods to an implementation for you, and some you can add yourself.

The adapter works through the Adapter API. To use Adapter API methods in adapter source code:

- Use the Adapter Designer to specify implementation information. The Adapter Designer adds the appropriate Adapter API methods to the adapter's Java source file when it generates the implementation.
- Add methods to Java code produced by the Adapter Designer.

The adapter modules and sample and utility adapters contain Adapter API methods and custom Java code.

Note: In this chapter, an "empty" business object means that no field values are set. The business object has been created, but the field values are null. You can use the Business Object API to set values. Remember that an empty business object is invalid until its mandatory fields are set.

Adapter methods

An adapter is a subclass of the Adapter class. The following table is a summary of the methods in the Adapter interface. These methods are added to your adapter's Java source file based on the implementation information you entered in the Adapter Designer.

Important: If you need to create a new implementation using the Adapter Designer, and you added code to your existing implementation, save a backup copy of your current implementation file. The Adapter Designer doesn't preserve code you added to an existing implementation when it generates a new implementation. A good way to preserve your code is to use helper files for your adapter. See *Using helpers* on page 248.

The adapter type, implementation declaration, and Java source code must always be kept in sync. So, you need to let the Adapter Designer add the methods in the Adapter class for you, rather than adding them to the implementation yourself, unless you are absolutely sure that you can make changes that correspond to the information in the adapter type and implementation declaration—as the Adapter Designer would have generated the code. The adapter type and implementation declaration ensure that pertinent adapter information is available in Partner Agreement Manager windows and dialog boxes for the users that need it.

Use this method	To do this
checkForEvents	Create events that can start Partner Agreement Manager public processes. The Adapter Server calls this method to check for new events it needs to pass to Partner Agreement Manager. You set the polling frequency in the Adapter Manager separately for each adapter instance. To generate an event, you must write code in the body of the checkForEvents method to check for a condition in the business application, and add an event (and an associated variant or business object) to the event context by using the addVariantEvent or addBOElementEvent methods. The context is provided through the EventContext class. If you detect a lost connection during checkForEvents, you can throw an EndSystemNotAvailableException.
createTransaction Context	Provide a new TransactionContext instance. The default implementation returns null. Your adapter must override this method to return a TransactionContext implementation that contains calls to real TP Monitor functionality.
execute	Execute a specific operation. This method must be overridden by all adapters. In the body of this method, the Adapter Designer places calls to “dispatch” methods, where you can place code to implement the operations defined for this adapter. The execute method can be called by more than one thread in the Adapter Server. If the adapter supports transactions, the Adapter Server manages the transaction, committing after this method returns or rolling back if an exception is thrown. If the adapter does not support transactions, it must either commit or rollback all stateful actions performed during execution of this method. The context is provided through the OperationContext class. If you detect a lost connection during execute, you can throw an EndSystemNotAvailableException.

Use this method	To do this
reExecute	Handle the case where execute has already been called on this operation, but execution results were inconclusive due to process or computer failure. Adapters that can safely handle reexecution of stateful operations can override this method. The default implementation calls execute for stateless operations and throws an exception for stateful operations. The reExecute method can be called by more than one thread in the Adapter Server. The context is provided through the OperationContext class. If you detect a lost connection during reExecute, you can throw an EndSystemNotAvailableException.
shutdown	Perform any necessary cleanup before shutting down the adapter. In the body of this method, you can write code to perform any cleanup, such as disconnecting from a remote computer or closing database connections. You must clean up all resources allocated by the adapter before shutting down. Failure to do so can cause memory leaks or errors in any end system that you do not properly close connections to. If this method is not in your adapter code, your adapter cannot perform adapter-specific tasks at shutdown. Do not throw any exceptions in the shutdown method.
startup	Initialize the adapter (including loading properties) when you start the adapter instance in the Adapter Manager. The startup method is also called when the Adapter Server starts, if you configured the adapter instance to be started at Adapter Server startup. This method lets you perform adapter-specific tasks at startup. In the body of this method, you must include code to perform any initialization, such as connecting to an end system such as a business application, database, or object request broker (ORB). All connections to end systems must be made in the startup method. If the adapter cannot connect successfully to the end system, the adapter needs to throw an EndSystemNotAvailableException. This exception is caught by the Adapter Server. If this adapter instance uses the auto-recovery feature, the Adapter Server will suspend the adapter and attempt to recover it. The Adapter Server attempts this recovery by retrying the call to the adapter's startup method until the adapter no longer throws an exception in startup. The auto-recovery properties that the Adapter Server uses are max recoveries and recovery interval. They are set in the error handling tab when you create the adapter instance. For information on setting error handling in the Adapter Manager, see the *Partner Agreement Manager Administrator's Guide*.

In addition, if you specify properties, the Adapter Designer adds the private loadAdapterProperties method to the startup call for you. As the name implies, this method loads the property values into your adapter. After the properties are loaded, you can access the object member variables directly; for example, this.user_name or this.password.

AdapterContext methods

The methods in the AdapterContext interface let you work with adapter properties. They are useful at startup and shutdown for working with property values set in the Adapter Manager. An adapter is not required to have properties.

You add properties from the Adapter Designer in the Properties panel of the Create Adapter Type dialog box. A property can be of the type string, integer, or boolean, be optional or mandatory (non-null), and can be encrypted (useful for passwords). In the Adapter Designer, you can restrict a property to a set of allowable values and set default values.

You set property values in the Adapter Manager, individually for each adapter instance. Each adapter instance can have unique property values that can distinguish it from other adapter instances of the same adapter type, implementation declaration, and implementation. You cannot set values while the adapter is started; it must be stopped.

Note: An empty string ("") as a data value is equivalent to null. A string containing one or more spaces (" ") is not null, and is considered to be data.

The following table is a summary of the AdapterContext interface methods. Unless otherwise noted, the Adapter Designer doesn't add the AdapterContext method to your adapter for you.

Use this method	To do this
getBoundProperties	Get the names of all properties that have values.
getDataType	Identify the type of data contained in a property that has a value. Returns one of the constants STRING, INTEGER, or BOOLEAN.
getProperties	Get the names of all properties declared in the adapter definition, whether or not they have a value.

Use this method	To do this
getPropertyAsBoolean	Get the value of a property as a boolean. This method is added by the Adapter Designer for each boolean property you specified for an adapter.
getPropertyAsInt	Get the value of a property as an integer. This method is added by the Adapter Designer for each integer property you specified for an adapter.
getPropertyAsString	Get the value of a property as a string. Depending on the data type, this method will do the following: integer: Returns the result of String.valueOf(int) string: Returns the string property value boolean: Returns the result of String.valueOf(boolean) This method is added by the Adapter Designer for each string property you specified for an adapter.
isPropertyBound	Determine if a property has a value.
isPropertyDeclared	Determine if there is a declared property with the specified name.

EventContext methods

A checkForEvents call can contain code to generate events based on a condition in the business application; these events are passed to Partner Agreement Manager to start public processes. The following table is a summary of the methods in the EventContext interface, which you can use within the body of a checkForEvents call. The Adapter Designer doesn't add these methods to an implementation for you.

Use this method	To do this
addBOElementEvent	Add an event and a business object to the EventContext object associated with the checkForEvents call. The empty business object is passed back to the adapter to get values.
addVariantEvent	Add an event and string data to the EventContext object associated with the checkForEvents call.

Use this method	To do this
getTransactionContext	Get the transaction context for this execution. You use this method if you need to make your checkForEvents call transactional.
removeEvent	Remove an event from the context. This method is useful for error handling when you are working with complex business objects, for example. When the adapter encounters an error condition while populating a business object, you can roll back the event.

ExecutionID methods

The following table is a summary of the ExecutionID interface methods. You can use execution, private process, and operation IDs to keep track of different tasks your adapter is performing, both in code and as a record in a database. (You can get operation IDs through the OperationContext interface.) The Adapter Designer doesn't add these methods to an implementation for you.

Use this method	To do this
getDisplayName	Get a human-readable description of the execution ID.
getID	Get a globally unique execution ID string for an operation execution. It is never null.
getPrivateProcessID	Get the private process ID of the private process that executed this operation.

OperationContext methods

You can use the methods in the OperationContext interface to work with operations. The methods can appear in the body of the execute methods generated for a specific operation, and in the reExecute method. The following table is a summary of the OperationContext interface methods. Unless otherwise noted, the Adapter Designer doesn't add these methods for you.

Use this method	To do this
createOutputBOElement	Create an empty output business object of the correct type for a particular Get or Post operation. This method returns the object, so any changes to the business object are in the context. The Adapter Designer adds this to adapter code if you specified an output business object for an operation.
createStatusBOElement	Create an empty status business object of the correct type for a Get or Post operation. This method returns the object, so any changes are persisted in the context. The Adapter Designer adds this to adapter code if you specified an output status business object for an operation.
getExecutionID	Get the unique execution ID for this operation execution. This ID can be used by adapters to match reexecute and undo calls to the original execution. The method returns an ExecutionID instance.
getExecutionMode	Get the mode of execution for this operation: TEST or PRODUCTION. This mode corresponds to the mode of the public process initiating the operation. Adapters that support distinct test and production behavior can check this mode and act accordingly.
getInput	Get the Object value for a specific input context variable. The input is either a business object or a string, depending on whether the input type is business object or variant. You can use this method as a generic way to get input values, if you don't know the type of input.
getInputBOElement	Get the business object values for a Post operation or for another business object you specified. The Adapter Designer adds this method for each input business object you specified.
getInputNames	Get the names of all input context variables for a particular operation, as defined in the Adapter Designer. It returns an enumeration of input name strings. Remember that an input might not have a value if it's optional.

Use this method	To do this
getInputVariant	Get the value of a variant input to an operation. It returns a string. The Adapter Designer adds this method for each input variant you specified.
getOperationID	Get a string that identifies which adapter operation is being executed. The ID is unique for all operations of the same type, for example, “Get purchase order.” The Adapter Designer adds this method to an implementation for you.
getOperationTypeID	Get a constant that identifies the operation type: GET, POST, or ADVANCED. The Adapter Designer adds this method to an implementation for you.
getOutputBOElement	Get an output business object of an operation.
getOutputVariant	Get the value of an output variant of an operation. It returns a string.
getStatusBOElement	Get the status business object of a Get or Post operation.
getTransactionContext	Get the transaction context for this execution. This will be a context instance generated by the createTransactionContext method of the adapter. You use this method in your transactional call to the business application. The Adapter Server uses the transaction context: if the adapter supports transactions, the Adapter Server manages the transaction, committing after this method returns or rolling back if an exception is thrown.
isFirstExecution	Determine if this is the first time this operation has been executed. This method is useful if you are using the requestRetry method, for example, because you might want to do some tasks during the initial operation execution request.
isReexecuted	Determine if the operation is being reexecuted (through the reExecute method). For example, you might want to code different behavior if an operation is reexecuted.
isStateful	Determine if this operation has been designated by the Adapter Designer as stateless (it does not modify the state of external applications). If an operation is stateful, you might need to code a rollback or other compensatory functionality into your error handling.
log	Log a message that will be inserted into the Partner Agreement Manager audit log for the process executing this operation.

Use this method	To do this
requestRetry	Specify that the operation cannot be completed on this call and needs to be attempted again. The same OperationContext will be passed into the execution method on the retry (any outputs set during previous execution(s) will be preserved).
setOutputVariant	Set the value of an output variant for an operation. The value is a string. If there is an existing value, it is overwritten.
unsetOutput	Unset the value of a variant or business object output of an operation.
unsetOutputBO	Unset the values of the output business object of a Get or Post operation. An Advanced operation can have more than one output, so you cannot use this method with an Advanced operation.
unsetStatusBO	Unset the values of a status business object for a Get or Post operation. An Advanced operation has no explicit status business object, so you cannot use this method with it.

TransactionContext methods

The following table is a summary of the methods in the TransactionContext interface, which you use in conjunction with the createTransactionContext call to implement transactional operations:

Use this method	To do this
begin	Start a new transaction.
commit	Commit a transaction.
rollback	Roll back a transaction.

About Exceptions

There are two exceptions the Adapter API methods throw:

- ISException — a generic exception.
- EndSystemNotAvailableException — indicates your adapter cannot successfully connect to a business system. This is sub-class of ISException.

ISException

ISException is a generic exception that you can use to signal failures within the Adapter Server. For example, the Adapter Designer adds the following code to an adapter, which you can add to:

```
/**
* Called when the adapter is started for the first time.
* @param context Holds Adapter specific properties, may be
* empty.
* @exception ISException Thrown if there is a problem starting
the Adapter.
*/

public void startup(AdapterContext context)
  throws ISException {

// load the properties from the adapter
loadAdapterProperties(context);

// add startup code here
}
```

For a more detailed explanation of why a call failed, examine the exception string message. Nested exception reporting is supported (when several exceptions occur, an ISException can contain an exception). Localized messages, in the end-user’s language (like Spanish or Italian), are also supported.

Here are the declarations:

```
public ISException(String localized_message,
  Throwable nested-exception)
public ISException()
public ISException(String message)
public ISException(Throwable nested-exception)
```

See *Implementing exception handling* on page 256 for more information.

Following is an example of what you can put after “add startup code here:”

```
try {
  //some operation in business application
} catch(Exception e) {
  throw new ISException("Error in business application", e);
}
```

e is a nested exception.

If you don’t handle an exception, the process aborts.

EndSystemNotAvailableException

When the startup method of your adapter fails to connect to an end system such as a database or business application, throw the specific exception EndSystemNotAvailableException, rather than a generic ISException. It is good coding practice to use specific exceptions when they are available as they allow you to handle each exception appropriately. EndSystemNotAvailableException is caught by the Adapter Server. If the adapter instance uses the auto-recovery feature, the Adapter Server suspends the adapter and retries it by calling its startup method until the adapter no longer throws an exception in startup. For more on setting error handling in an adapter instance, see the *Partner Agreement Manager Administrator’s Guide*.

Executing operations

When a private process runs and reaches an Extension action, the Adapter Server calls the execute method of an adapter. The adapter then performs the specific operation that the private process needs.

The Adapter Designer gets you started by helping you define the inputs, outputs, and method names for each operation, which appear as Java code in the implementation file.

You can perform the following operations by using an adapter:

- *Get* information from a business application—without changing the state of the business application (a stateless operation). The Get operation must have an output business object and optionally an additional status business object. It can have zero or more input context variables (business objects, variants, or both); each input can be mandatory or optional (null at run time). The output business object type tells the business application what information to return.
- *Post* data to the business application (insert, update, or delete), and return output and status data to Partner Agreement Manager. The state of the business application changes: it is a stateful operation. The Post operation must have one populated input business object, and optionally return an output business object, status business object, or both. The output can provide information about the state of the business application through a transaction ID, for example.
- Perform a complex operation (called an *Advanced* operation) that cannot be implemented with one Get or Post. It can be stateful or stateless. It can have zero or more input context variables, and zero or more output context variables (business objects, variants, or both). Each input and output can be optional or mandatory.

To handle the reexecution of operations that didn't complete due to process or computer failure, you can implement the reExecute method in your adapter.

The OperationContext interface holds the context for the execute or reExecute call, and provides methods relevant to operations, such as getting input values and setting output values. The methods can appear in the body of the execute methods generated for a specific operation, and in the reExecute method.

Adapter class

An adapter is a subclass of the Adapter class. These methods are added to your adapter's Java source file based on the implementation information you entered in the Adapter Designer.

If you need to initialize your adapter on startup, you can add code to the body of the startup method. If you need to perform any cleanup on adapter shutdown, you can specify in the Adapter Designer that you want a shutdown method in your adapter implementation. Typically an adapter connects with an business application. The startup method is the place to put the code to create this connection. Likewise, the shutdown method is where you must close that connection and free all resources allocated by your adapter. Failure to free allocated resources can result in memory leaks.

checkForEvents method

The Adapter Server calls this method to check for new events it needs to pass to Partner Agreement Manager. (You set the polling frequency when you add the adapter instance in the Adapter Manager.) The adapter must override this method to implement event generation functionality. To generate an event, you must write code in the body of the checkForEvents method to:

- check for a condition in the business application
- add a variant or business object to the event context by using the addVariantEvent or addBOElementEvent methods

You might use a variant if you need to pass a scalar variable to a public process. You might use a business object when you need structured information.

If you specify one or more event types in the Events panel of the Create Adapter Type dialog box, the Adapter Designer adds an empty checkForEvents method to the implementation, which you can then complete. It also creates a constant for each event type, based on the name it had in the Adapter Designer. You use the constant when you write code in the body of the checkForEvents method to generate the event based on some condition in the business application. Typically, your code reads information from the business application to determine if an event needs to be generated.

Let the Adapter Designer add the checkForEvents method, rather than adding it to the implementation yourself, to keep the adapter type, implementation declaration, and implementation in sync. The adapter type and implementation declaration ensure that events are visible in Partner Agreement Manager windows and dialog boxes for the users that need it. The events appear in the Process Manager window (under Partner > Processes > Events), where process designers can associate a public process with an event so when Partner Agreement Manager receives the event, it starts that process.

IMPORTANT: If you need to create a new implementation using the Adapter Designer, and you added code to your existing implementation, save a backup copy of your current implementation file. The Adapter Designer doesn't preserve code you added to an existing implementation when it generates a new implementation. A good way to preserve your code is to use helper files for your adapter. See *Using helpers* on page 248.

The checkForEvents method can perform stateful operations against the business application. In that case, you'd use the transaction context for your stateful method call.

Declaration

public void checkForEvents(EventContext context) throws ISException

Parameters

context functions as a container for produced events.

Return value

none

Exceptions

Thrown if there is an error checking for the event.

You can also throw EndSystemNotAvailableException if you detect a lost connection during the checkForEvents.

Examples

In the following example, the Adapter Designer added the checkForEvents call, and the programmer added the code in bold to the implementation. A user specified an "Example inventory item below reorder quantity" event type in the Adapter Designer, so the Adapter designer also added a constant for this event type. This constant is used in the code the programmer wrote for generating an event of this type. The code determines if a single item is below reorder quantity, which can generate one event per checkForEvents call. The checkForEvents call returns a variant event. See *Java implementation example* on page 257 for more information.

```
//-------------------------------------------------------------
// Constants - Protected - Adapter Events
//-------------------------------------------------------------

protected static final String
  EVENT_EXAMPLE_INVENTORY_ITEM_BELOW_REORDER_QUANTITY =
  "Example inventory item below reorder quantity";

...

public final void checkForEvents(EventContext context)
  throws ISException {

  // add event production code here

  // XXX added for TestingInventoryAdapter - begin

  // trap for case when no item has been created yet
  if (this.item_number != Integer.MIN_VALUE
      && this.quantity_on_hand < this.reorder_quantity) {
          context.addVariantEvent(
           EVENT_EXAMPLE_INVENTORY_ITEM_BELOW_REORDER_QUANTITY,
           Integer.toString(this.item_number),
           null);
  }

  // XXX added for TestingInventoryAdapter - end
}
```

Next is alternate code where checkForEvents returns a business object event instead of a variant event. The only changes are a new local variable and the contents within the if construct. However, remember that the adapter type must reflect the change in the event type.

```
public final void checkForEvents(EventContext context)
  throws ISException {

  Element event_bo; // XXX added for TestingInventoryAdapter

  // add event production code here

  // XXX added for TestingInventoryAdapter - begin

  // trap for case when no item has been created yet
  if (this.item_number != Integer.MIN_VALUE
      && this.quantity_on_hand < this.reorder_quantity) {
          event_bo = context.addBOElementEvent(
           EVENT_EXAMPLE_INVENTORY_ITEM_BELOW_REORDER_QUANTITY,
           null);
          event_bo.setData("item_number",
              Integer.toString(this.item_number));
          event_bo.setData("description", this.description);
          event_bo.setData("quantity_on_hand",
              Integer.toString(this.quantity_on_hand));
          event_bo.setData("warehouse_id", this.warehouse_id);
  }

  // XXX added for TestingInventoryAdapter - end
}
```

See also

addBOElementEvent, addVariantEvent, checkForEvents, createTransactionContext, execute, getTransactionContext, removeEvent, reExecute, shutdown, startup

createTransactionContext method

This method is a callback to provide a new TransactionContext instance. The default implementation returns null. Your adapter must override this method to return a TransactionContext implementation that contains calls to real TP Monitor functionality. So you would have another Java class that calls the TP Monitor functionality.

If you select Create Transaction Context in the Advanced panel of the Generate Java Source Code dialog box, the Adapter Designer adds an empty createTransactionContext method to the implementation, which you can then complete.

Declaration

public TransactionContext createTransactionContext()
throws ISException

Parameters

none

Return value

TransactionContext instance

Exceptions

Thrown if unable to create a TransactionContext, for example, due to some exception in the class that implements the commercial TP Monitor functionality.

Example

Before operation execution, you need to implement a class that wraps calls to a commercial TP Monitor:

```
public class MyTC extends TransactionContext {
  private TPMonitor tp_monitor

  public MyTC() {
  tp_monitor = new TPMonitor();
  }
  public void begin() {
      // do database BEGIN TRANSACTION
      // tp_monitor.begin_trans();
  }

  public void commit() {
      // add commit code
      // tp_monitor.rollback_trans();
  }

  public void rollback() {
      // add rollback code
  }

  TransactionContext createTransactionContext() throws
ISException {
      MyTC tc = new MyTC();
      return((TransactionContext) tc);
  }
}
```

See also

begin, checkForEvents, commit, execute, reExecute, rollback, shutdown, startup

EXECUTE METHOD

The Adapter Server calls this method to execute a specific operation, in response to the Extension action in a private process that is running. All adapters must override this method, even if the adapter does not handle operations, because it's an abstract method.

IMPORTANT: If you need to create a new implementation using the Adapter Designer, and you added code to your existing implementation, save a backup copy of your current implementation file. The Adapter Designer doesn't preserve code you added to an existing implementation when it generates a new implementation.

If you specify operations in the Operations panel of the Create Adapter Type dialog box, the Adapter Designer adds an execute method to the implementation, as well as methods based on the operation names you specified in the Adapter Designer, and generates the dispatch code of the execute method. You can then complete the empty methods for the operations you want the adapter to perform. Let the Adapter Designer add the execute method for you, unless you are absolutely sure that you can make changes that keep the adapter type, implementation declaration, and implementation in sync. The adapter type and implementation declaration ensure that pertinent adapter information is available in Process Manager windows and dialog boxes for the users that need it.

You can use asynchronous callbacks with the execute method. If you use an asynchronous callback, you must use the requestWaitForCallback method. If you do this and execute throws an ISException or EndSystemNotAvailableException, execute is treated as synchronous.

The execute method can be called by several threads in the Adapter Server. If your adapter is not thread-safe, select the Synchronize Operation Execute Methods option in the Advanced panel of the Generate Java Source Code dialog box of the Adapter Designer (it's selected by default). This makes the execute methods for specific operations thread-safe, because no more than one thread can operate at a time; however, it's slower than allowing several threads to call several methods at the same time. Here is an example of a thread-safe execute method (as indicated by the synchronized keyword) for an operation:

```
protected synchronized void
executeGetForecastForAYear(OperationContext context)
```

If the adapter supports transactions, the Adapter Server manages the transaction, committing after this method returns or rolling back if an exception is thrown. If the adapter doesn't support transactions, it must either commit or roll back all stateful actions performed during execution of this method.

WARNING: If you don't implement transactions and a computer executing an operation fails, such as from a loss of power, your business application can be left in an indeterminate state.

DECLARATION

public abstract void execute(OperationContext context)
throws ISException

PARAMETERS

context is the run-time context for the operation. It is a container for operation inputs and outputs (output values must be added by the adapter implementation).

RETURN VALUE

none

EXCEPTIONS

Thrown if there is an error during execution, such as an error condition in the business application, network problem, out of memory error, invalid business object, no read permission on a file, and so on. If you are interacting with another business application, you can use ISException to return an appropriate error message. You can also throw EndSystemNotAvailableException if you detect a lost connection during execution.

EXAMPLE

The following example is from the sample code in *Java implementation example* on page 257. The Adapter Designer adds all of the following code, based on the operations specified in the Adapter Designer:

- One Get: Get Inventory Information
- Two posts: Create Inventory Item and Update Item Quantity
- No Advanced operations

```
public void execute(OperationContext context)
  throws ISException {
      int operation_type_id;
      String operation_id;

      operation_type_id = context.getOperationTypeID();
      operation_id = context.getOperationID();

      switch (operation_type_id) {
          case OperationContext.GET:
          if (operation_id.equals(GET_INVENTORY_INFORMATION)) {
              executeGetInventoryInformation(context);
              }else {
              throw new ISException(UNKNOWN_OPERATION_ID + ": " +
              operation_id);
              }
              break;

          case OperationContext.POST:
              if (operation_id.equals(CREATE_INVENTORY_ITEM)) {
              executeCreateInventoryItem(context);
              }else if
(operation_id.equals(UPDATE_ITEM_QUANTITY)) {
              executeUpdateItemQuantity(context);
              }else {
              throw new ISException(UNKNOWN_OPERATION_ID + ": " +
              operation_id);
              }
              break;

          case OperationContext.ADVANCED:
          break;

          default:
              throw new ISException(UNKNOWN_OPERATION_TYPE_ID +
": " +
              operation_type_id);
      }
}
```

See also

checkForEvents, createTransactionContext, reExecute, shutdown, startup

reExecute method

The Adapter Server calls this method when execute has already been called on this operation, but execution results were inconclusive due to process or computer failure. If you are coding your adapter to handle the reexecution of stateful operations, you need to override this method. As with the execute method, this method can be called by several threads in the Adapter Server, so you need to make your code thread-safe.

If you select ReExecute in the Advanced panel of the Generate Java Source Code dialog box, the Adapter Designer adds an empty reExecute method to the implementation, which you can then complete.

The default implementation of the reExecute() method depends on if the operation is stateful or stateless. If the operation is stateless, the default implementation will just call execute(). If the operation is stateful, the default implementation will throw an ISException, which will result in erroring the extension action in the private process. The default implementation is given as an example.

Declaration

public abstract void reExecute(OperationContext context) throws ISException

Parameters

context is the run-time context for the operation. It is the container for operation inputs and outputs (output values must be added by the adapter implementation).

Return value

none

Exceptions

Thrown if there is an error reexecuting the operation.

You can also throw EndSystemNotAvailableException if you detect a lost connection during reexecution.

Example

```
public void reExecute(OperationContext context)
       throws ISException {
  if (!context.isStateful()) {
      execute(context);
  } else {
      throw new ISException("Re-execution isn’t supported for" +
                  " stateful operations.");
  }
}
```

See also

checkForEvents, createTransactionContext, execute, shutdown, startup

SHUTDOWN METHOD

When you stop an adapter instance in the Adapter Manager, the Adapter Server calls the shutdown method. In the body of this method, you need to write code to perform any necessary cleanup, such as releasing resources, disconnecting from a remote computer, or closing database connections before the adapter terminates. Failure to release allocated resources before adapter termination can result in memory leaks. If this method isn't in your adapter code, your adapter cannot perform adapter-specific tasks at shutdown.

If you select Shutdown in the Advanced panel of the Generate Java Source Code dialog box, the Adapter Designer adds an empty shutdown method to the implementation, which you can then complete.

IMPORTANT: If you need to create a new implementation using the Adapter Designer, and you added code to your existing implementation, save a backup copy of your current implementation file. The Adapter Designer doesn't preserve code you added to an existing implementation when it generates a new implementation. A good way to preserve your code is to use helper files for your adapter. See *Using helpers* on page 248.

DECLARATION

public void shutdown(AdapterContext context) throws ISException

PARAMETERS

context holds adapter-specific properties, if any.

RETURN VALUE

none

EXCEPTIONS

Do not throw an exception.

Example

In the following example, the Adapter Designer added the shutdown method and the programmer added the line in bold, which calls shutdown on the helper class. See *Java implementation example* on page 257 for more information.

```
public void shutdown(AdapterContext context)
  throws ISException {

  // add shutdown code here

  // XXX added for InventoryAdapter
  this.helper.shutdown();
}
```

PAM API

Adapter API, Adapter abstract class

See also

checkForEvents, createTransactionContext, execute, getBoundProperties, getDataType, getProperties, getPropertyAsBoolean, getPropertyAsInt, getPropertyAsString, isPropertyBound, isPropertyDeclared, reExecute, startup

STARTUP METHOD

When you start an adapter instance in the Adapter Manager, the Adapter Server calls the startup method. The startup method is also called when the Adapter Server starts, if you configured the adapter instance to be started at Adapter Server startup. In the body of this method, you can write code to perform any initialization, such as connecting to a business application, database, or object request broker (ORB). If the connection to the business application fails, throw EndSystemNotAvailableException. If you set error handling in the adapter instance, this exception causes the Adapter Server to suspend the adapter and attempt to restart it in case the error connecting to the business system was momentary. You can use the AdapterContext to get property values for use by the adapter instance during startup and shutdown.

The Adapter Designer always adds an empty startup method to the implementation, which you can then complete. If you do not add code to this method, your adapter doesn't perform adapter-specific tasks at startup.

In addition, if you specify properties, the Adapter Designer adds the private loadAdapterProperties method to the startup call for you. As the name implies, this method loads the properties and values into your adapter.

IMPORTANT: If you need to create a new implementation using the Adapter Designer, and you added code to your existing implementation, save a backup copy of your current implementation file. The Adapter Designer doesn't preserve code you added to an existing implementation when it generates a new implementation. A good way to preserve your code is to use helper files for your adapter. See *Using helpers* on page 248.

DECLARATION

public void startup(AdapterContext *context*) throws ISException

PARAMETERS

context holds adapter-specific properties, if any.

RETURN VALUE

none

EXCEPTIONS

Thrown if there is a problem starting the adapter.

EndSystemNotAvailableException is available for use when the attempt to connect to the business system fails. This exception can be caught by the Adapter Server, resulting in the temporary suspension of the adapter. The Adapter Server then attempts to restart the adapter. For more on setting error handling for an adapter instance, see the *Partner Agreement Manager Administrator's Guide*.

Example

In the following example, the Adapter Designer added the startup method and the loadAdapterProperties method, and a programmer added the line in bold, which does JDBC-specific adapter initialization to connect to a database. See *Java implementation example* on page 257 for more information.

```
public void startup(AdapterContext context)
  throws ISException {

  // load the properties from the adapter
  loadAdapterProperties(context);

  // add startup code here

  // XXX added for InventoryAdapter
  this.helper.startup(this.jdbc_driver, this.database_url,
      this.user, this.password, this.reorder_quantity);
}
```

See also

checkForEvents, createTransactionContext, execute, getBoundProperties, getDataType, getProperties, getPropertyAsBoolean, getPropertyAsInt, getPropertyAsString, isPropertyBound, isPropertyDeclared, reExecute, shutdown

AdapterContext interface

The methods in the AdapterContext interface let you work with adapter properties. They are useful at startup and shutdown for working with property values set in the Adapter Manager.

You set property values in the Adapter Manager, individually for each adapter instance. Each adapter instance can have unique property values that distinguish it from other adapter instances of the same adapter type, implementation declaration, and implementation. You cannot set values while the adapter is started; the adapter must be stopped.

You set property values in the Adapter Manager, individually for each adapter instance. The instance must be in a stopped state before you can specify property values.

getBoundProperties method

This method gets the names of all properties that have values. The Adapter Designer does not add this method to an implementation for you.

Note: An empty string ("") as a data value is equivalent to null. A string containing one or more spaces (" ") is not null, and is considered to be data.

Declaration

public abstract Enumeration getBoundProperties()

Parameters

none

Return value

Enumeration of string property names, if any.

Exceptions

none

Example

The following example gets the names of all properties that have been set. Optional properties might not have values.

```
Enumeration enum = adapter_context.getBoundProperties();
while (enum.hasMoreElements()) {
  System.out.println("Property " + (String) enum.nextElement()+
              " is set.");
}
```

See also

getDataType, getProperties, getPropertyAsBoolean, getPropertyAsInt, getPropertyAsString, isPropertyBound, isPropertyDeclared, shutdown, startup

getDataType method

This method identifies the type of data contained in a property that has a value. The Adapter Designer does not add this method to an implementation for you.

Declaration

int getDataType(String name) throws ISException

Parameters

name is the property name, which cannot be null.

Return value

One of the constants STRING, INTEGER, or BOOLEAN

Exceptions

Thrown if the specified name is null or if there is no bound property with that name.

Example

The following example shows how to use getDataType.

```
int type = adapter_context.getDataType("jdbc_url");
switch (type) {
case AdapterContext.STRING:
   String val = adapter_context.getPropertyAsString();
   break;
case AdapterContext.BOOLEAN:
   boolean val = adapter_context.getPropertyAsBoolean();
   break;
case AdapterContext.INTEGER:
   int val = adapter_context.getPropertyAsInt();
   break;
default:
   throw new ISException("Unknown type: " + type);
}
```

See also

getBoundProperties, getProperties, getPropertyAsBoolean, getPropertyAsInt, getPropertyAsString, isPropertyBound, isPropertyDeclared, shutdown, startup

getProperties method

This method gets the names of all properties declared in the adapter definition, whether or not they have a value. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract Enumeration getProperties()

Parameters

none

Return value

Enumeration of string property names, if any.

Exceptions

none

Example

The following example gets the names of all properties whether they have been set or not.

```
Enumeration enum = adapter_context.getProperties();
while (enum.hasMoreElements()) {
  System.out.println("Property " + (String) enum.nextElement() +
             " is defined.");
}
```

See also

getBoundProperties, getDataType, getPropertyAsBoolean, getPropertyAsInt, getPropertyAsString, isPropertyBound, isPropertyDeclared, shutdown, startup

getPropertyAsBoolean method

This method is an accessor for boolean properties. It is added by the Adapter Designer for each boolean property you specified for an adapter.

Declaration

public abstract boolean getPropertyAsBoolean(String *name*)
throws ISException

Parameters

name is the name of the property. It cannot be null.

Return value

Property value as a boolean.

Exceptions

Thrown if the specified name is null, if the specified property is not of type boolean, or if there is no bound property with the specified name.

Example

The following example shows how to use getPropertyAsBoolean.

```
int type = adapter_context.getDataType("jdbc_url");
switch (type) {
case AdapterContext.STRING:
   String val =
adapter_context.getPropertyAsString("jdbc_url");
   break;
case AdapterContext.BOOLEAN:
   boolean val =
adapter_context.getPropertyAsBoolean("jdbc_url");
   break;
case AdapterContext.INTEGER:
   int val = adapter_context.getPropertyAsInt("jdbc_url");
   break;
default:
   throw new ISException("Unknown type: " + type);
}
```

See also

getBoundProperties, getDataType, getProperties, getPropertyAsInt, getPropertyAsString, isPropertyBound, isPropertyDeclared, shutdown, startup

getPropertyAsInt method

This method is an accessor for integer properties. It is added by the Adapter Designer for each integer property you specified for an adapter.

Declaration

public abstract int getPropertyAsInt(String *name*) throws ISException

Parameters

name is the name of the property. It cannot be null.

Return value

Property value as an integer.

Exceptions

Thrown if the specified name is null, if the specified property is not of type integer, or if there is no bound property with the specified name.

Example

In the following code, if the property identified by the constant PROP_REORDER_QUANTITY has a value, then the value is assigned to the variable reorder_quantity. (The Adapter Designer creates constants based on the properties you specified.) See *Java implementation example* on page 257 for the complete example adapter.

```
if (context.isPropertyBound(PROP_REORDER_QUANTITY)) {
  reorder_quantity =
          context.getPropertyAsInt(PROP_REORDER_QUANTITY);
}
```

Another example showing how to use getPropertyAsInt.

```
int type = adapter_context.getDataType("jdbc_url");
switch (type) {
case AdapterContext.STRING:
   String val =
adapter_context.getPropertyAsString("jdbc_url");
   break;
case AdapterContext.BOOLEAN:
   boolean val =
adapter_context.getPropertyAsBoolean("jdbc_url");
   break;
case AdapterContext.INTEGER:
   int val = adapter_context.getPropertyAsInt("jdbc_url");
   break;
default:
   throw new ISException("Unknown type: " + type);
}
```

See also

getBoundProperties, getDataType, getProperties, getPropertyAsBoolean, getPropertyAsString, isPropertyBound, isPropertyDeclared, shutdown, startup

getPropertyAsString method

This method gets the value of a property as a string. It is added by the Adapter Designer for each string property you specified for an adapter.

Declaration

public abstract int getPropertyAsIString(String *name*) throws ISException

Parameters

name is the name of the property. It cannot be null.

Return value

Property value as a string. Depending on the data type of a property, this method will do one of the following:

- integer: Returns the result of String.valueOf(int)
- string: Returns the string property value
- boolean: Returns the result of String.valueOf(boolean)

Exceptions

Thrown if the specified name is null, if there is no property with the specified name, or if the property value is null.

Example

The following example gets the value of a user name property. See *Java implementation example* on page 257 for the complete example adapter.

```
user = context.getPropertyAsString(PROP_USER);
```

Another example:

```
int type = adapter_context.getDataType("jdbc_url");
switch (type) {
case AdapterContext.STRING:
   String val =
adapter_context.getPropertyAsString("jdbc_url");
   break;
case AdapterContext.BOOLEAN:
   boolean val =
adapter_context.getPropertyAsBoolean("jdbc_url");
   break;
case AdapterContext.INTEGER:
   int val = adapter_context.getPropertyAsInt("jdbc_url");
   break;

default:
   throw new ISException("Unknown type: " + type);
}
```

See also

getBoundProperties, getDataType, getProperties, getPropertyAsBoolean, getPropertyAsInt, isPropertyBound, isPropertyDeclared, shutdown, startup

isPropertyBound method

This method determines if a property has a value. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract boolean isPropertyBound(String *name*)

Parameters

name is the name of the property. If the name is null, the method returns false.

Return value

Boolean true if the property has a value; boolean false if no property exists with the specified name, if the property has no value, or if the specified name is null.

Exceptions

none

Example

In the following code, if the optional property named `PROP_REORDER_QUANTITY` has a value, then the value is assigned to the variable reorder_quantity. See *Java implementation example* on page 257 for the complete example adapter.

```
if (context.isPropertyBound(PROP_REORDER_QUANTITY)) {
  reorder_quantity =
          context.getPropertyAsInt(PROP_REORDER_QUANTITY);
}
```

See also

getBoundProperties, getDataType, getProperties, getPropertyAsBoolean, getPropertyAsInt, getPropertyAsString, isPropertyDeclared, shutdown, startup

isPropertyDeclared method

This method determines if there is a declared property with the specified name. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract boolean isPropertyDeclared(String *name*)

Parameters

name is the name of the property. If it is null, the method returns boolean false.

Return value

Boolean true if there is a declared property; boolean false if there is no declared property with the specified name or if the specified name is null.

Example

The following example tests if a property named “jdbc_driver” is declared.

```
if (adapter_context.isPropertyDeclared("jdbc_driver")) {
   System.out.println("Property jdbc_driver is declared for
            this adapter");
}
```

See also

getBoundProperties, getDataType, getProperties, getPropertyAsBoolean, getPropertyAsInt, getPropertyAsString, isPropertyBound, shutdown, startup

EventContext interface

A checkForEvents call can contain code to generate events based on a condition in the business application; these events are passed to Partner Agreement Manager to start public processes. You can use the methods in the EventContext interface within the body of a checkForEvents call. The Adapter Designer does not add these methods to an implementation for you.

addBOElementEvent method

This method adds an event and a business object to the EventContext object associated with the checkForEvents call. The empty business object is passed back to the adapter to get values. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract Element addBOElementEvent(String *type*, int *mode*, String *desc*) throws ISException

public abstract Element addBOElementEvent(String *type*, String *desc*) throws ISException

Parameters

type is the event type name, which is a constant generated by the Adapter Designer based on the event type name as it appeared in the Adapter Designer. It cannot be null.

mode (optional) is one of the constants `TEST`, `PRODUCTION`, or `UNSPECIFIED`. If no mode is specified, it defaults to the constant `UNSPECIFIED`. If you specify `TEST`, only test versions of a public process are started with this event; if you specify `PRODUCTION`, only production versions are started; if you use `UNSPECIFIED`, both test and production versions can be started. (Remember that you specify whether a process is test or production in the Process Manager.)

desc is a description of the event source, for example, "SAP R3." It can be null. You can use the description to help with logging and auditing.

Return value

Empty Element business object for the event. You can pass it to the removeEvent method to eliminate this event from the context, if needed.

Exceptions

Thrown if there is an error adding the event, such as a null, empty, or invalid type.

Example

In the following example, an event and business object are generated, then data is added to the business object:

```
public final void checkForEvents(EventContext context)
  throws ISException {

  Element event_bo; // XXX added for TestingInventoryAdapter

  // add event production code here

  // XXX added for TestingInventoryAdapter - begin

  // trap for case when no item has been created yet
  if (this.item_number != Integer.MIN_VALUE
      && this.quantity_on_hand < this.reorder_quantity) {
          event_bo = context.addBOElementEvent(
           EVENT_EXAMPLE_INVENTORY_ITEM_BELOW_REORDER_QUANTITY,
           null);
          event_bo.setData("item_number",
              Integer.toString(this.item_number));
          event_bo.setData("description", this.description);
          event_bo.setData("quantity_on_hand",
              Integer.toString(this.quantity_on_hand));
          event_bo.setData("warehouse_id", this.warehouse_id);
  }

  // XXX added for TestingInventoryAdapter - end
}
```

See also

addVariantEvent, checkForEvents, getTransactionContext, removeEvent

addVariantEvent method

This method adds an event and string data to the EventContext object associated with the checkForEvents call. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract Object addVariantEvent(String *type*, String *data*, int *mode*, String *desc*) throws ISException

public abstract Object addVariantEvent(String *type*, String *data*, String *desc*) throws ISException

Parameters

type is the event type name, which is a constant generated by the Adapter Designer based on the event type name as it appeared in the Adapter Designer. It cannot be null or empty ("").

data is the string data associated with the event. It is null for an event with no data.

mode (optional) is one of the constants `TEST`, `PRODUCTION`, or `UNSPECIFIED`. If no mode is specified, it defaults to the constant `UNSPECIFIED`. If you specify `TEST`, only test versions of a public process are started with this event; if you specify `PRODUCTION`, only production versions are started; if you use `UNSPECIFIED`, both test and production versions can be started. (Remember that you specify whether a process is test or production in the Process Manager.)

desc is a description of the event source, for example, "SAP R3." It can be null. You can use the description to help with logging and auditing.

Return value

Handle to the event, which you can pass into the removeEvent method to eliminate this event from the context, if needed.

Exceptions

Thrown if there is an error adding the event, such as an invalid type.

Example

In the following example, the checkForEvents call returns an event and string data. See *Java implementation example* on page 257 for the complete example adapter.

```
public final void checkForEvents(EventContext context)
  throws ISException {

  // add event production code here

  // XXX added for TestingInventoryAdapter - begin

  // trap for case when no item has been created yet
  if (this.item_number != Integer.MIN_VALUE
      && this.quantity_on_hand < this.reorder_quantity) {
          context.addVariantEvent(
           EVENT_EXAMPLE_INVENTORY_ITEM_BELOW_REORDER_QUANTITY,
           Integer.toString(this.item_number),
           null);
  }

  // XXX added for TestingInventoryAdapter - end
}
```

See also

addBOElementEvent, checkForEvents, getTransactionContext, removeEvent

getTransactionContext method

This method gets the transaction context for this execution. Use this method if you need to make your checkForEvents call transactional. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract TransactionContext getTransactionContext()

Parameters

none

Return value

Instance of TransactionContext.

Exceptions

none

Example

The following example gets the generic transaction context and casts it to a known subclass.

```
MyTransactionContext tc = (MyTransactionContext)
context.getTransactionContext();
```

See also

addBOElementEvent, addVariantEvent, checkForEvents, createTransactionContext, removeEvent

removeEvent method

This method removes an event from the context. It is useful for error handling when you are working with complex business objects, for example. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract boolean removeEvent(Object *o*)

Parameters

o is the Event handle— an Object that was returned by the addVariantEvent method, or an Element business object that was returned by the addBOElementEvent method.

Return value

Boolean true if the event was successfully removed; boolean false if there was no event with that handle.

Exceptions

none

Example

The following example shows how to use removeEvent.

```
Element elem = event_context.addBOElementEvent(bo);

//If we can't set the BO's data properly, then remove
//the event from the context...
if (!setBOData(bo)) {
   event_context.removeEvent(elem);
}
```

See also

addBOElementEvent, addVariantEvent, checkForEvents, getTransactionContext

ExecutionID interface

You can use execution, private process, and operation IDs to keep track of different tasks your adapter is performing, both in code and as a record in a database. For example, in a reExecute method, you can use a switch statement to identify an operation by execution ID, operation ID, or both. The Adapter Designer does not add these methods to an implementation for you.

An *execution ID* is unique to a private process and an operation.

A *private process ID* is unique to a run of a private process that started an operation.

An *operation ID* is the name of an operation in the Adapter Designer and is unique to an operation type in an implementation; if the same operation runs more than once, they have the same ID. The *operation type ID* specifies whether the operation is Get, Post, or Advanced. See *Java implementation example* on page 257 for more information on getting these IDs.

getDisplayName method

This method gets a human-readable description of the execution ID. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract String getDisplayName()

Parameters

none

Return value

Description string of execution ID you can display.

Exceptions

none

Example

The following example prints out a human readable form of the execution ID.

```
System.out.println("Execution ID: " +
exec_id.getDisplayName());
```

See also

getExecutionID, getID, getPrivateProcessID

getID method

This method gets a globally unique execution ID string for an operation execution. It is never null. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract String getID()

Parameters

none

Return value

Execution ID string (not human-readable)

Exceptions

none

Example

The following example stores the ID in a database.

```
helper.persistID(context.getExecutionID().getID());
```

See also

getDisplayName, getExecutionID, getPrivateProcessID

getPrivateProcessID method

This method gets the private process ID of the private process that executed this operation. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract String getPrivateProcessID()

Parameters

none

Return value

Private process ID string, which is never null.

Exceptions

none

Example

The following example stores the private process in a database.

```
helper.persistPrivProcID(context.getExecutionID().getPrivatePro
cessID());
```

See also

getDisplayName, getExecutionID, getID

OperationContext interface

You can use the methods in the OperationContext interface to work with operations. The methods can appear in the body of the execute methods generated for a specific operation, and in the reExecute method. Unless otherwise noted, the Adapter Designer does not add these methods for you.

createOutputBOElement method

This method returns an empty output business object of the correct type for a specific Get or Post operation. The Adapter Designer adds this to adapter code if you specified an output business object for an operation.

Declaration

public abstract Element createOutputBOElement(String *output*) throws ISException

public abstract Element createOutputBOElement() throws ISException

Parameters

output (optional) is the output name. It cannot be null.

Return value

Empty Element business object instance that you can work with.

Exceptions

Thrown if there is an error creating the business object, if the specified output was not defined in the adapter type, or if the specified output name is null.

Example

In the following example, the output and status business objects are created from the operation context. The programmer can now use the Business Object API with these business objects through the output_bo and status_bo variables. See *Using Business Object API methods* on page 191 for information on the BusinessObject API. Also, see *Java implementation example* on page 257 for the complete example adapter.

```
// create the output Business Objects
output_bo = context.createOutputBOElement();
status_bo = context.createStatusBOElement();
```

See also

createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

createStatusBOElement method

This method returns an empty status business object of the correct type for a specific Get or Post operation. The Adapter Designer adds this to adapter code if you specified an output status business object for an operation.

Declaration

public abstract Element createStatusBOElement() throws ISException

Parameters

none

Return value

Empty Element business object instance.

Exceptions

Thrown if there is an error creating the business object or if this operation is not of type Get or Post.

Example

In the following example, the output and status business objects are created from the operation context. The programmer can now use the Business Object API with these business objects through the output_bo and status_bo variables. See *Java implementation example* on page 257 for the complete example adapter.

```
// create the output Business Objects
output_bo = context.createOutputBOElement();
status_bo = context.createStatusBOElement();
```

See also

createOutputBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getExecutionID method

This method gets the unique execution ID for this operation execution. This ID can be used by adapters to match reexecute and undo calls to the original execution. The method returns an ExecutionID instance. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract ExecutionID getExecutionID()

Parameters

none

Return value

An object implementing the ExecutionID interface, which is never null.

Exceptions

none

Example

The following example retrieves the execution ID from the context, then uses it to log the ID into a database:

```
try {
  ExecutionID id = context.getExecutionID();
  logTransaction(id); //logs this execution ID to a database
} catch(SQLException e) {
  throw new ISException("Unable to log transaction", e);
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getExecutionMode method

This method gets the mode of execution for this operation: `TEST` or `PRODUCTION`. This mode corresponds to the mode of the public process that initiated the operation. Adapters can check the mode if they need to act differently based on test or production versions. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract int getExecutionMode()

Return value

One of the constants `TEST` or `PRODUCTION`.

Example

For example, you can have the following code in the execute method so you connect to the test or production system as appropriate:

```
try {
  int mode = context.getExecutionMode();
  if(mode==OperationContext.TEST) {
      //run operation against the test system
      executeTransaction("TEST");
  } elseif(mode==OperationContext.PRODUCTION) {
      //run operation against the production system
      executeTransaction("PRODUCTION");
  }
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getInput method

This method gets the Object value for a specific input context variable, either a business object or variant. You can use this method as a generic way to get input values, if you don't know the type of input. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract Object getInput(String *input*) throws ISException

Parameters

input is a constant created by the Adapter Designer for the input context variable name you specified in the Adapter Designer.

Return value

Input value. Null if the input has not been bound.

Exceptions

Thrown if the specified input does not exist or if the specified input name is null.

Example

The following example tests whether the context_var1 input is a business object or variant.

```
Object input = getInput("context_var1");
if (input instanceof Element) {
   //It's a business object
   Element tmp = (Object) input;
   String f1 = tmp.getData("field1");
} else if (input instanceof String) {
   //It's a variant
   String var1 = (String) input;
} else {
   // Error case
   throw new ISException("Unsupported input type: " +
input.getClass());
}
```

SEE ALSO

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getInputBOElement method

This method gets the business object values for a Post operation or for another business object you specified. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract Element getInputBOElement() throws ISException

public abstract Element getInputBOElement(String *input*)
throws ISException

Parameters

input (optional) is the input business object name, identified by a constant that the Adapter Designer created in your Java file. If it is null, the method returns null. If you do not specify an input, the method gets the input of a Post operation.

Return value

The Element business object.

Exceptions

Thrown if this operation is not of type Post. If you specified an *input*, an exception is thrown if the specified input does not exist, if the input data is not an Element business object, or if the specified input name is null.

Example

In the following example, the method gets the value of the business object that was input to a Post operation (the input is stored in the context). See *Java implementation example* on page 257 for the complete example adapter.

```
// load the inputs from the OperationContext
Element input_bo = context.getInputBOElement();
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getInputNames method

This method gets the names of all input context variables you defined in the Adapter Designer for a particular operation. Although not often needed, this method is useful if you need metadata information about the operation. Remember that an optional input might not have values. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract Enumeration getInputNames()

Parameters

none

Return value

Enumeration of input name strings, specified as constants generated by the Adapter Designer based on the names you entered there. Empty if there are no inputs.

Exceptions

none

Example

The following example prints out the names of all the inputs that have been set.

```
Enumeration names = context.getInputNames();
while (name.hasMoreElements()) {
   String input = (String) names.nextElement();
   System.out.println("Input: " + input);
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getInputVariant method

This method gets the value of a variant input to an operation. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract String getInputVariant(String *input*)
throws ISException

Parameters

input is a constant created by the Adapter Designer for the input variant you specified in the Adapter Designer. For example, if the operation was called "my Post operation" and the input variant was called "my_variant", the constant the Adapter Designer adds would be MY_POST_OPERATION_INPUT_MY_VARIANT. If the input is null, the method returns null.

Return value

The variant value, as a string. Returns null if the variant has no value.

Exceptions

Thrown if the specified variant does not exist, if the variant value is not a string, or if the specified variant name is null.

Example

In the following example, the method gets the value of the item_number input to a "Get inventory information" operation, represented by the constant GET_INVENTORY_INFORMATION_INPUT_ITEM_NUMBER (created by the Adapter Designer). See *Java implementation example* on page 257 for the complete example adapter.

```
// load the inputs from the OperationContext
String item_number =
context.getInputVariant(GET_INVENTORY_INFORMATION_INPUT_ITEM_NU
MBER);
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getOperationID method

This method gets a string that identifies which adapter operation is being executed. The ID is unique for all operations of the same type, for example, “my Post Operation.” The Adapter Designer adds this method to an implementation for you.

Declaration

public abstract String getOperationID()

Return value

An operation ID string, which is never null.

Example

The following example contains code the Adapter Designer generates for you. See *Java implementation example* on page 257 for the complete example adapter.

```
public void execute(OperationContext context)
  throws ISException {
      int operation_type_id;
      String operation_id;

      operation_type_id = context.getOperationTypeID();
      operation_id = context.getOperationID();
...
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getOperationTypeID method

This method gets a constant that identifies the operation type: GET, POST, or ADVANCED. The Adapter Designer adds this method to an implementation for you.

Declaration

public abstract int getOperationTypeID()

Return value

One of the constants `GET`, `POST`, or `ADVANCED`.

Example

The following example contains code that the Adapter Designer generates for you. See *Java implementation example* on page 257 for the complete example adapter.

```
public void execute(OperationContext context)
  throws ISException {
      int operation_type_id;
      String operation_id;

      operation_type_id = context.getOperationTypeID();
      operation_id = context.getOperationID();
...
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getOutputBOElement method

This method gets the output Element business object of an operation. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract Element getOutputBOElement()

public abstract Element getOutputBOElement(String *output*)

Parameters

output (optional) is the output name, which is a constant created by the Adapter Designer based on the name you entered there. If it is null, the method returns null. You specify an output if you know you might have more than one output (as can be the case for an Advanced operation).

Return value

Output of the operation. It is null if the output hasn't been set, the operation isn't of type Get or Post, the output doesn't have an Element business object value, the specified output doesn't exist for this operation, or if the specified output name is null.

Example

Code like the following is frequently used to set output business object data within adapter implementations:

```
Element output_bo = context.getOutputBOElement();
output_bo.setData(FIELD1, "your value");
output_bo.setData(FIELD2, "your value");
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getOutputVariant method

This method gets the value of an output variant of an operation. It returns a string. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract String getOutputVariant(String *output*)

Parameters

output is the output name, which is a constant created by the Adapter Designer based on the name you entered there. If it is null, the method returns null.

Return value

Output value. Null if the output is either not set or doesn't have a String value.

Example

The following example prints out the value of an output variant.

```
String val = context.getOutputVariant("return_code");
System.out.println("Return code = " + return_code);
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getStatusBOElement method

This method gets the status business object of a Get or Post operation. (Advanced operations don't explicitly have status business objects, although you can add them.) The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract Element getStatusBOElement()

Parameters

none

Return value

Status Element business object of the operation. Null if the status hasn't been set or if the operation is not of type Get or Post.

Example

The following example prints out the value of the StatusBO.

```
Element elem = context.getStatusBOElement();
System.out.println("StatusBO.result: " +
elem.getData("result");
System.out.println("StatusBO.resason: " +
elem.getData("reason");
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

getTransactionContext method

This method gets the transaction context for this execution. This will be a context instance generated by the createTransactionContext method of the adapter. If you select the Create Transaction Context option in the Advanced panel of the Create Java Source Code dialog box, the Adapter Designer adds this method to an implementation for you.

Declaration

public abstract TransactionContext getTransactionContext()

Return value

Instance of TransactionContext, or null if the adapter doesn't support transactions.

Example

The following example gets the generic transaction context and casts it to a known subclass.

```
MyTransactionContext tc = (MyTransactionContext)
context.getTransactionContext();
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

isFirstExecution method

This method determines if this is the first time this operation has been executed. This method is useful if the requestRetry method is being used (your code might depend on whether it is a first execution). The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract boolean isFirstExecution()

Return value

Boolean true if this is the first execution; boolean false if this is a retry.

Example

The following example is the requestRetry() execute method implementation

```
protected void executeInterfaceTable(OperationContext context)
   throws ISException {

  Element input_bo;
  Element output_bo;
  Element status_bo;

  // load the inputs from the OperationContext
  input_bo = context.getInputBOElement();

  // create the output Business Objects
  output_bo = context.createOutputBOElement();
  status_bo = context.createStatusBOElement();

  //Test if it's the first time execute() has been called; if
  //so, then perform a database INSERT. For subsequent times,
  //check if the end system has updated the database table's
  //status from PENDING to COMPLETE.
  if (context.isFirstExecution()) {
     //Perform an INSERT into the interface table
     String id = helper.insertIntoInterfaceTable();

     //Store the id in the output BO
     output_bo.setData("id", id);
  } else {
      //Check if the batch program has run
      if (helper.checkBatchStatus(output_bo.getData("id")) ==
              COMPLETE) {
          output_bo.setData("status", "COMPLETE");
      } else {
          //Tell the AS to retry every 10 minutes since we aren't
          // yet done
          context.requestRetry(600);
      }
  }
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

isReexecuted method

This method determines if the operation is being reexecuted (through the reExecute method). The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract boolean isReexecuted()

Return value

Boolean true if the operation is being reexecuted; boolean false otherwise.

Example

The following example checks the state of an operation.

```
if (context.isReexecuted()) {
  //Check state of this operation
  if (checkOperationState(context.getExecutionID()) ==
              COMPLETED) {
      status_bo.setData("result", "success");
      status_bo.setData("reason", "Previous execution was
              successful");
  } else {
      // Previous execution was unsuccessful; go do it
      executeMyMethod(status_bo);
  }
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

isStateful method

This method determines if this operation has been designated by the Adapter Designer as stateless (it doesn't modify the state of external applications). If an operation is stateful, you might need to code a rollback or other compensatory functionality into your error handling. This call is useful in a reExecute method. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract boolean isStateful()

Return value

Boolean true if the operation is stateful; boolean false otherwise.

Example

The following example shows how to use isStateful.

```
public void reExecute(OperationContext context)
      throws ISException {
  if (!context.isStateful()) {
      execute(context);
  } else {
      throw new ISException("Re-execution isn't supported for" +
                  " stateful operations.");
  }
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

LOG METHOD

This method logs a message that will be inserted into the Partner Agreement Manager audit log for the process executing this operation. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract boolean log(int *severity*, String *message*)

public abstract boolean log(int *severity*, String *message*, Throwable *t*)

Parameters

severity is one of the constants ERROR, WARNING, or INFO.

message is a string description of the error. If null, no log will be added.

t (optional) is the throwable object associated with the error, which can be null. If supplied, the exception and its stack trace is written to the audit log.

Return value

True if the log was successfully added.

Example

The following example logs the attempt to connect to MQSeries.

```
context.log(OperationContext.INFO, "Initiating connection to
MQSeries...");
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

operationCompleted method

This method finishes an asynchronous operation, including error-handling. The Adapter Designer does not add this method to an implementation for you.

Note: The start to an asynchronous call is in the execute method.

A call to operationCompleted will result in an error if the asynchronous call returns before requestWaitForCallback has finished. In this case, the call is regarded as synchronous.

Declaration

public void operationCompleted(ISException *t*) throws ISException

Parameters

t is the throwable object associated with an error as a result of this operation. which can be null. When *t* is null, it means that the asynchronous operation finished without error.

Exceptions

Exceptions are thrown is the asynchronous call cannot be completed. Since the asynchronous call cannot throw an exception in the same way as a synchronous operation, a parameter is used.

ISException is thrown if more than one callback is made for an operation. The first callback will be processed. All subsequent ones will be ignored and operationCompleted will throw an ISException.

ISException is also thrown if the callback happens after the operation times out.

If either the asynchronous operation or the asynchronous callback fails because of an EndSystemNotAvailableException, both the operation and the adapter instance are suspended. If the Adapter Server is configured to do so, it attempts to restart the adapter. If that is successful, it will rerun the operation. If you use this functionality, you must roll back all unfinished work before throwing EndSystemNotAvailableException. Otherwise, the results of the operation will be indeterminate. If the Adapter Server is not configured to restart the adapter, an EndSystemNotAvailableException is treated as an ISException. Timeout values are not reset for adapter restarts and operation retries.

Example

The following example completes an asynchronous operation.

```
context.operationCompleted(t);
```

See also

requestWaitForCallback

requestRetry method

This method specifies that the operation cannot be completed on this call and needs to be attempted again. The same OperationContext will be passed into the execution method on the retry (any outputs set during previous execution(s) will be preserved). You use this method to implement long-running operations. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract void requestRetry(int *seconds*)

Parameters

seconds is the number of seconds that the Adapter Server waits to call execute on the adapter again for this operation to find out if it has completed. The value must be between 1 and 2419200 (four weeks).

Example

```
protected void executeInterfaceTable(OperationContext context)
   throws ISException {
  Element input_bo;
  Element output_bo;
  Element status_bo;

  // load the inputs from the OperationContext
  input_bo = context.getInputBOElement();

  // create the output Business Objects
  output_bo = context.createOutputBOElement();
  status_bo = context.createStatusBOElement();

  //Test if it's the first time execute() has been called; if
  //so, then perform a database INSERT. For subsequent times,
  //check if the end system has updated the database table's
  //status from PENDING to COMPLETE.
  if (context.isFirstExecution()) {
      //Perform an INSERT into the interface table
      String id = helper.insertIntoInterfaceTable();

      //Store the id in the output BO
      output_bo.setData("id", id);
  } else {
      //Check if the batch program has run
      if (helper.checkBatchStatus(output_bo.getData("id")) ==
              COMPLETE) {
          output_bo.setData("status", "COMPLETE");
      } else {
          //Tell the IS to retry every 10 minutes since we aren't
          //yet done
          context.requestRetry(600);
      }
  }
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO, operationCompleted, requestWaitForCallback

requestWaitForCallback method

This method registers this operation for asynchronous handling. The Adapter Designer does not add this method to an implementation for you.

Note: The start to an asynchronous call is in the execute method.

If the asynchronous call returns before requestWaitForCallback has finished, the call to operationCompleted will result in an error. In this case, the call is regarded as synchronous.

Declaration

public void requestWaitForCallback() throws ISException

Exceptions

Thrown if the call was not properly processed by the Adapter Server.

Example

The following example registers this operation for asynchronous handling.

```
context.requestWaitForCallback();
```

See also

operationCompleted

setOutputVariant method

This method sets the value of an output variant for an operation. The value is a string. If there is an existing value, it is overwritten. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract void setOutputVariant(String *output*, String *value*)
throws ISException

Parameters

output is the name of the output argument, which is a constant generated by the Adapter Designer based on the name you entered there. It cannot be null.

value is the string value; use null to unset.

Exceptions

Thrown if the specified output variant doesn't exist, if the specified output is not of type variant, or if the specified output name is null.

Example

The following example sets a variant called "transaction_id" in the private process context to the value of **tid**.

```
context.setOutputVariant("transaction_id", tid);
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO, unsetStatusBO

unsetOutput method

This method unsets the value of a variant or business object output of an operation. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract void unsetOutput(String *output*) throws ISException

Parameters

output is the name of the output argument, which is a constant generated by the Adapter Designer from the name you entered there. It cannot be null.

Exceptions

Thrown if the specified output doesn’t exist, or if the specified output name is null.

Example

This method is useful when some error condition is reached.

```
try {
   //Call the end system and get the transaction id
   String tid = middleware_handle.send(msg);

   //Set the transaction id into the context variant
   context.setOutputVariant("transaction_id", tid);
//Set the transaction id into the Output BO
   output_bo.setData("trx_id", tid);

   //Set the Status BO
   status_bo.setData("result", "success");
   status_bo.setData("reason", "failure");

   //Try to send an ack back to the middleware system
   middleware_handle.sendAck(tid);
} catch (IOException e) {
   //Since the ack failed, we need to unset the variant and
   //the two BOs we set in the try block above.
   context.unsetOutput("transaction_id");
   context.unsetOutputBO();
   context.unsetStatusBO();
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutputBO, unsetStatusBO

unsetOutputBO method

This method unsets the values of the output business object of a Get or Post operation. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract void unsetOutputBO() throws ISException

Exceptions

Thrown if this operation is not of type Get or Post.

Example

This method is useful when some error condition is reached.

```
try {
   //Call the end system and get the transaction id
   String tid = middleware_handle.send(msg);

   //Set the transaction id into the context variant
   context.setOutputVariant("transaction_id", tid);
//Set the transaction id into the Output BO
   output_bo.setData("trx_id", tid);

   //Set the Status BO
   status_bo.setData("result", "success");
   status_bo.setData("reason", "failure");

   //Try to send an ack back to the middleware system
   middleware_handle.sendAck(tid);

} catch (IOException e) {
   //Since the ack failed, we need to unset the variant and
   //the two BOs we set in the try block above.
   context.unsetOutput("transaction_id");
   context.unsetOutputBO();
   context.unsetStatusBO();
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetStatusBO

unsetStatusBO method

This method unsets the values of a status business object for a Get or Post operation. The Adapter Designer doesn't add this method to an implementation for you.

Declaration

public abstract void unsetStatusBO() throws ISException

Exceptions

Thrown if this operation is not of type Get or Post.

Example

This method is useful when some error condition is reached.

```
try {
   //Call the end system and get the transaction id
   String tid = middleware_handle.send(msg);

   //Set the transaction id into the context variant
   context.setOutputVariant("transaction_id", tid);
//Set the transaction id into the Output BO
   output_bo.setData("trx_id", tid);

   //Set the Status BO
   status_bo.setData("result", "success");
   status_bo.setData("reason", "failure");

   //Try to send an ack back to the middleware system
   middleware_handle.sendAck(tid);

} catch (IOException e) {
   //Since the ack failed, we need to unset the variant and
   //the two BOs we set in the try block above.
   context.unsetOutput("transaction_id");
   context.unsetOutputBO();
   context.unsetStatusBO();
}
```

See also

createOutputBOElement, createStatusBOElement, execute, getExecutionID, getExecutionMode, getInput, getInputBOElement, getInputNames, getInputVariant, getOperationID, getOperationTypeID, getOutputBOElement, getOutputVariant, getStatusBOElement, getTransactionContext, isFirstExecution, isReexecuted, isStateful, log, reExecute, requestRetry, setOutputVariant, unsetOutput, unsetOutputBO

TransactionContext interface

You might need to make operations fail-safe against data corruption from power failures and so on. In this case, you can incorporate Transaction Processing Monitor (TP Monitor) calls into your adapter code. Use the Adapter API methods described in this section if you want to implement transaction processing.

BEGIN METHOD

This method starts a new transaction. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract void begin() throws ISException

Exceptions

Thrown for an error in starting a transaction, such as not being able to connect to the TP Monitor or a network resource.

See also

commit, createTransactionContext, rollback

COMMIT METHOD

This method commits a transaction. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract void commit() throws ISException

Exceptions

Thrown for an error in committing a transaction, such as not being able to connect to the TP Monitor or a network resource.

See also

begin, createTransactionContext, rollback

ROLLBACK METHOD

This method rolls back a transaction. The Adapter Designer does not add this method to an implementation for you.

Declaration

public abstract void rollback() throws ISException

Exceptions

Thrown for an error in committing a transaction, such as not being able to connect to the TP Monitor or a network resource.

See also

begin, commit, createTransactionContext

CHAPTER
9 Using Business Object API methods

Read this chapter for information on the business object API methods that you can use in your adapter code. This chapter will give Java programmers the information necessary to expand on the Java implementation created by the Adapter Designer. The business object (BO) API is in the com.extricity.document.api package.

This chapter includes these sections:

About the Business Object API

The Business Object API has two interfaces that an adapter developer will use: Element and ElementSequence. The Element interface provides methods for working with group and field elements, and the ElementSequence interface provides methods that work with sequences of elements.

Group vs. field elements

Partner Agreement Manager requires that every element consist of data only *or* one or more sub-elements. That is, an element can be the root of an element subtree, or it can contain data. An element might not directly contain both data and sub-elements.

Elements that contain sub-elements are referred to as group elements; elements that contain data only are referred to as field elements. This interface includes methods for operating on both group elements and field elements. Note that some methods are defined to work only on either a group or a field element and will throw an exception if the element is not of the expected type.

Note: Some methods are defined to operate over all data contained in the Element—whether it is a field element, or a group element containing data located in more than one subtree. Some examples are clearData() and hasData().

Data

All data must be set using a string representation of the data unit. Likewise, data might be retrieved only in its string form. A data type String is available to describe the data that might be contained in a field element. Partner Agreement Manager supports only one data type, Java.lang.String.

Copying

All methods that take an element reference as a parameter copy any data objects contained in that parameter element (see copyIn). The source element must be of the same element type as the receiver element, but it need not be a member of the same business object instance.

Tag path strings

Several of the methods in this interface take a tag path string as a parameter. This string is used to identify and operate on an element or a sequential set of elements of the same type (an ElementSequence).

A tag path indicates a position in the element tree relative to this element. An element identified by a tag path will either be this element itself, if the path is null or an empty string, or a descendant element. It's not possible to identify a parent or ancestor element with a tag path.

For example, given some group element, the path string *Header* would identify the child element of that group with tag name *Header*. The path string *Header/Company_Name* would identify the *Company_Name* element contained in the "Header" child group element.

Note: Tag paths are case-sensitive. If you want to access a child element of type *Header*, you must use the path *Header* not *header*.

Because field elements cannot contain child elements, an InvalidQueryException will always be thrown if a tag path method is called on a field element and the path string is not null or an empty string.

An indexing notation is supported to allow access to a descendant element that is repeatable. For example, if a group element named Purchase_Order is defined to contain a sequence of Line_Item child elements, a path string accessing a specific Line_Item element *must* use the index notation to identify one of the Line_Item children.

```
line_item = po.getElement("Line_Item[0]")
```

This code gets the first Line_Item element that occurs in the element po. If the index value is less than zero or greater than or equal to the number of elements in the sequence accessed, an IndexOutOfBoundsException will be thrown.

An ElementSequence might be retrieved by omitting the index in the last term of a tag path string. Such a path might only be used in the getElementSequence method. For example,

```
line_item_seq = po.getElementSequence("Line_Item")
```

This code gets an ElementSequence that contains the elements of type Line_Item that occur within the **po** element.

To address an attribute of an element, use the @ notation. For example, foo@bar is the attribute bar of the element foo.

BusinessObject methods

The BusinessObject interface represents the entire BusinessObject. It contains information about the type of the business object as well as access to the root of the content tree. In addition, it provides methods for serializing and de-serializing via an XML character stream.

Use this BusinessObject method	To do this
deepClone	Create a new business object of the same type, with the same data.
fromStream	Deprecated. Use readStream instead.
fromXMLString	Initialize the content of this business object from the specified XML String
getDocument	Return the W3C Document for manipulation using the W3C DOM API.
getRootElement	Get the root element of the content tree.
getTypeID	Get the type identifier for this BusinessObject.
getTypeURI	Deprecated. Use getTypeID instead.
readStream	Initialize the content of this business object from the specified stream.
toStream	Deprecated. Use writeStream instead.
toXMLString	Initialize the content of this business object from the specified XML String
validate	Validate this business object.
writeStream	Write the content of this BusinessObject to a stream.

Element methods

The methods in the Element interface let you access and manipulate the content of a business object instance. Both fields and groups are elements: fields are elements that can contain data, and groups are elements that contain other elements. The following is a summary of the Element interface methods that are currently available:

Use this Element method	To do this
clearAll	Deprecated. Do not use this method. Use removeElement instead.
clearData	Deprecated. Do not use this method. Use removeElement instead.
copyIn	Copy data from an element to another element of the same business object type, either within the same business object or between business objects. The elements copied from and to must be of the same element type: you can copy a group into a group, and a field into a field, and they must have a hierarchy of subordinate elements that is a subset of the hierarchy being copied into. Subordinate element sequences are copied; the element sequence length doesn't have to be the same between elements.
getAttr	Get the value of the specified attribute for this element.
getAttrDefault	Return the default value for this attribute.
getBusinessObject	Get the business object that "owns" this element instance.
getData	Return a string representation of the data contained in a field.
getElement	Get a reference to the group or field identified by a tag path string. The tag path can specify an element in a sequence by its index number.
getElementSequence	Get a reference to an element sequence. An element sequence is a collection of consecutive "sibling" elements of the same element type. The group or field is specified as repeatable in the group that contains it.
getTagName	Get the name of this element. This can also be thought of as the element type name.
hasData	Check whether a group or field contains data (is not null). A zero length string is considered null.

Use this Element method	To do this
isField	Check whether an element is a field or group. This procedure is useful when you want to use code that can manipulate different business object types, for example.
isValid	Determine the validity of an element based on the content of the business object type.
removeAttr	Remove the attribute from this element.
removeAttrs	Remove all attributes from this element.
removeElement	Remove this element.
setAttr	Set the specified attribute for this element.
setData	Set the data contained in a field element. This method can also be used to set attribute values.
toString	Return a string describing the validity and content of an element and any elements subordinate to it that contain data. Required subordinate elements appear in the description string; optional subordinate elements without any data do not appear. This helps you see which elements must have data for this element to be valid. You can use the println method to display the value returned by toString.

ElementSequence methods

The methods in the ElementSequence interface let you manipulate an element sequence: a collection of consecutive Element objects of the same element type. Element sequences are indexed starting at zero (0); for example, four occurrences of an element (the sequence length is 4) will be indexed 0, 1, 2, 3. Valid index values are zero to the length of the sequence minus one (0 to *length* - 1).

The following is a summary of the ElementSequence interface methods that are currently available:

Use this ElementSequence method	To do this
getElementAt	Get the group or field at the specified position in this element sequence.
hasData	Check whether any element in an element sequence contains data.
length	Return the number of elements in this element sequence. This is useful for setting boundary values to loop through all the elements in an element sequence.
newElement	Add a new element to the end of this sequence and return the newly created element.
newElementAt	Insert a new element at the specified position in the sequence and return a reference to the newly created element. Adds 1 to the index of the element currently in that position (if any) and any following elements, so they are "shifted to the right." As with the newElement procedure, you can use this procedure to add to the end of the sequence.
removeAll	Remove all elements in a sequence. Any data contained in any of the elements is deleted. The length of this element sequence becomes 0.
removeElementAt	Remove an element (and its data) at the specified position in a sequence. The indexes of elements at greater index values are reduced by 1 (they are "shifted to the left"). The length of the sequence is reduced by 1.

BusinessObject interface

The BusinessObject interface contains information about the type of the business object as well as access to the root of the content tree. In addition, it provides methods for serializing and deserializing via an XML character stream.

deepClone method

Creates a new business object from this business object. The new business object has the same type and data as the original business object. The new business object will share no mutable information with this business object.

Declaration

BusinessObject deepClone()

Return value

BusinessObject is the new, cloned business object.

fromStream method

This method has been deprecated. Use readStream instead.

fromXMLString method

Initializes the content of this business object from the specified XML String.

An element that has an empty value or is not represented is considered to have a value of the empty string. Likewise, if mixed content is not present for an element that supports mixed content, the value of that element will be set to the empty string.

Declaration

void fromXMLString(String *str*) throws IOException

Parameters

str is the XML String, corresponding to an instance of this type of business object.

Exceptions

IOException is thrown if the String is not correct XML.

See Also

writeStream, from XMLString

getDocument method

Return the W3C Document for manipulation using the W3C DOM API.

Declaration

org.w3c.dom.Document getDocument()

Return value

Document, the W3C document.

getRootElement method

Gets the root Element of the content tree.

Declaration

Element getRootElement()

Return value

Element, the root element of the content tree.

getType ID method

Gets the type identifier for this business object.

Declaration

BOTypeID getTypeID()

Return value

BOTypeID, the type ID of this business object.

getTypeURI method

This method has been deprecated. Use getTypeID instead.

readStream method

Initializes the content of this business object from the specified stream.

An element that has an empty value or is not represented is considered to have a value of the empty string. Likewise, if mixed content is not present for an element that supports mixed content, the value of that element will be set to the empty string.

Declaration

void fromStream(Reader *r*) throws IOException

Parameters

r is the reader, the XML character stream corresponding to an instance of this type of business object.

Exceptions

IOException is thrown if there is a problem reading the stream.

See Also

writeStream, fromXMLString, toXMLStream

toStream method

This method has been deprecated. Use writeStream instead.

toXMLString method

Writes the content of this business object to a string.

The returned string contains an XML declaration and a DOCTYPE entity. The value of the SYSTEM ID string used in the DOCTYPE might be either the BODefID or an external ID.

Declaration

String toXMLString(boolean *use_external_id*)

Parameters

use_external_id indicates whether to use an external ID for the SYSTEM ID in the DOCTYPE, if one is available. If this is true, use an external ID. If false, don’t use one, use the BODefID instead.

Return value

Returns the business object in an XML string.

See Also

fromStream, fromXMLString, setAttr

writeStream method

Writes the content of this business object to the specified stream. An XML declaration and DOCTYPE is written. Unless specified, the SYSTEM ID is the BODefID.

Declaration

void writeStream(Writer *w*) throws IOException

void writeStream(OutputStream *o*) throws IOException

void writeStream(Writer *w*, boolean *use_external_id*, String *encoding*) throws IOException

void writeStream(OutputStream *o*, boolean *use_external_id*, String *encoding*) throws IOException

Parameters

w is the writer, the character stream that the contents of this business object will be written to in XML format.

o is the byte stream.

use_external_id indicates whether to use an external ID, if available for the SYSTEM ID.

encoding will be UTF-8 if this argument is null.

Exceptions

IOException is thrown if there is a problem writing to the stream.

See Also

readStream

VALIDATE METHOD

Validates this business object.

DECLARATION

String validate(Element *element*, boolean *deep*, boolean *null_and_empty_are_not_valid*)

PARAMETERS

element

deep

null_and_empty_are_not_valid indicates whether to consider null and empty elements as valid. True indicates that they are not valid, false that they are.

RETURN VALUE

String is either the first error or null if the business object is valid.

Element interface

The element interface provides programmatic access to the content of a business object. The logical structure of a business object's content is a tree of elements. Every instance of a business object has exactly one element that is the root of its element tree. Any element in a business object might be considered an “element subtree” of the business object's content. Every instance of an element exists within the context of a particular business object. Each element has a tag name, which is a string.

clearAll method

This method is deprecated. Use removeElement instead.

clearData method

This method is deprecated. Use removeElement instead.

copyIn method

This method copies an element to another element of the same business object type, either within the same business object or between business objects. The source and destination elements must be of the same element type.

- You can copy a group into a group, or a field into a field.
- The elements within a group must have a hierarchy that is a subset of the hierarchy being copied into.

Subordinate element sequences are also copied; after the copy, the element sequence length will be the same as the sequence copied from.

Declaration

void copyIn(Element *copy_element*) throws ElementTypeException

void copyIn(String *tag_path*, Element *copy_element*) throws InvalidQueryException, ElementTypeException, IndexOutOfBoundsException

Parameters

copy_element is the element you're copying from.

tag_path is a string indicating the location of the element or group in the element tree.

Exceptions

ElementTypeException indicates that the element types do not match. This exception is thrown if you try to copy to an element with a different tag name, for example, from po_data to po_number. Or you try to copy an element from a different business object type.

InvalidQueryException indicates that the tag path you supplied is invalid. For example, you can get this error if you typed a wrong name. This is also thrown if you specify an array index for a non-sequence, or if you didn't specify an array index for a sequence.

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

Example

This example copies the ship_to group in the first element of the po_line element sequence to the ship_to group of the second po_line element sequence:

```
Element line1_ship_to, line2_ship_to;

line1_ship_to = po.getElement("po_line[0]/ship_to");
line2_ship_to = po.getElement("po_line[1]/ship_to");
line2_ship_to.copyIn (line1_ship_to);
```

See Also

ElementTypeException, InvalidQueryException, IndexOutOfBoundsException

getAttr method

This method gets a string representation of the value contained in this attribute.

Declaration

String getAttr(String *attribute_name*) throws InvalidQueryException

Parameter

attribute_name is a string indicating the name of the attribute.

Return value

String representation of the value of this attribute, or an empty string if the value of this attribute hasn't been set.

Exceptions

InvalidQueryException indicates that the attribute name you supplied is invalid. For example, you can get this error if you typed a wrong name.

Example

This example returns the value of the attribute my_attribute. *value* holds the returned string.

```
String value;
value = po.getAttr("my-attribute");
```

See Also

setAttr, InvalidQueryException

getAttrDefault method

This method gets a string representation of the default value for this attribute. If no default value is specified, return null.

Declaration

String getAttrDefault(String *attribute_name*) throws InvalidQueryException

Parameter

attribute_name is a string indicating the name of the attribute.

Return value

String representation of the default value for this attribute, or null if no default value has been set.

Exceptions

InvalidQueryException indicates that the attribute name you supplied is invalid. For example, you can get this error if you typed a wrong name.

Example

This example returns the default value of the attribute my_attribute. *value* holds the returned string.

```
String value;
value = po.getDefaultAttr("my-attribute");
```

See Also

setAttr, getAttr, InvalidQueryException

getBusinessObject method

This method gets the business object that “owns” this element instance.

Important: Only use this method on the root element of the business object’s content tree.

Declaration

BusinessObject getBusinessObject()

Return value

BusinessObject that “owns” this element instance.

Exceptions

InvalidQueryException indicates that the attribute name you supplied is invalid. For example, you can get this error if you typed a wrong name.

Example

This example returns the value of the attribute my_attribute. *value* holds the returned string.

```
BusinessOBject bo = po.getBusinessObject();
bo.toStream(file_writer);
```

See Also

setAttr, InvalidQueryException

getData method

This method gets a string representation of the data contained in this field element or the field element identified by the tag path string.

Declaration

public abstract String getData() throws ElementTypeException

public abstract String getData(String *tag_path*) throws InvalidQueryException, ElementTypeException, IndexOutOfBoundsException

Parameter

tag_path is a string indicating the location of the element or group in the element tree.

Return value

String representation of the data in this field, or null if this field does not contain data. If the tag path is valid, but the element doesn't exist, the element is created. If the tag path specifies an attribute and value for the attribute is not set, the empty string is returned.

Exceptions

ElementTypeException indicates that you called this method on a group element instead of a field element. This is also thrown if you called this method on a group and that group does not support mixed content.

InvalidQueryException indicates that the tag path you supplied is invalid. For example, you can get this error if you typed a wrong name. This is also thrown if the data contains an invalid attribute. This is also thrown if you specify an array index for a non-sequence, or if you didn't specify an array index for a sequence.

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

Examples

This example returns the value of the item_code field in the first po_line group. *item* holds the returned string.

```
String item;
item = po.getElement("po_line[0]/item_code").getData();
```

This code does exactly the same thing.

```
String item;
item = po.getData("po_line[0]/item_code");
```

See Also

ElementTypeException, InvalidQueryException, IndexOutOfBoundsException

getDataType method

This method gets a string describing the type of data object that might be contained in this field element.

Declaration

public abstract String getDataType() throws ElementTypeException

Return value

String, data type, (String, Date, Integer,...). Only the String data type is supported.

Exceptions

ElementTypeException indicates that you called this method on a group element. This method only works on field elements.

Example

You can use the getDataType method to wrap the language-specific functionality, such as integer assignments in Java, from a String variable.

```
int ponum;
 if (po.getElement("po_number").getDataType().equals("String"))
  {
  ponum = new Integer(po.getData("po_number")).intValue();
  }
```

See Also

ElementTypeException

getElement method

This method gets a reference to the group or field identified by the tag path string. The tag path can specify either a nonrepeatable element or an element in an element sequence by using an index number. If the element doesn't exist, create it.

Declaration

Element getElement(String *tag_path*) throws InvalidQueryException, IndexOutOfBoundsException

Parameters

tag_path is a string indicating the path to the group or field, relative to *element*, which is a reference to a group or field.

Return value

Element, the element identified by the tag path.

Exceptions

InvalidQueryException indicates that the tag path you supplied is invalid. For example, you can get this error if you typed a wrong name. This is also thrown if you specify an array index for a non-sequence, or if you didn't specify an array index for a sequence.

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

Example

In this example, the ship_to group is assigned to the shipping_address variable:

```
Element shipping_address;
shipping_address = po.getElement("po_line[0]/ship_to");
```

See Also

InvalidQueryException, IndexOutOfBoundsException

getElementSequence method

This method gets a reference to an element sequence. An element sequence is a collection of consecutive sibling elements of the same element type. The group or field element identified by the tag path must be specified as repeatable by its parent group. An index must not be used with the last element in the tag path parameter.

If the specified element sequence does not exist, create it.

Declaration

ElementSequence getElementSequence(String *tag_path*) throws InvalidQueryException, IndexOutOfBoundsException

Parameters

tag_path is a string indicating the location of the element or group in the element tree.

Return value

ElementSequence.

Exceptions

InvalidQueryException indicates the tag path you supplied is invalid, or the tag path doesn't resolve to a sequence. This is also thrown if an array index is specified for the last component of the tag path.If an array index is specified for a parent that is not a sequence, this exception is thrown.

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

Example

This example assigns the po_line element sequence to the eseqLines variable (Purchase_Order defines po_line as repeatable):

```
ElementSequence eseqLines;
eseqLines = po.getElementSequence("po_line");
```

See Also

ElementSequence, InvalidQueryException, IndexOutOfBoundsException

getTagName method

This method gets the tag name of this element. This can also be thought of as the element type name.

Declaration

public abstract String getTagName()

Return value

tag name, which is a String indicating the name of this element. For example, a PO might have elements with tag names such as po_date and po_number.

Example

This example prints Purchase_Order:

```
System.out.println(po.getTagName());
```

hasData method

This method checks whether the element contains data.

- If the element is a field, the procedure returns true if the field contains data (is not null).
- If the element is a group, the procedure returns true if any subordinate field contains data.
- For an element sequence, the procedure returns true if any element in the sequence contains data.

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

Declaration

public abstract boolean hasData()

Return value

Boolean true, if this element contains data in any field, or any attribute is not null or " "; boolean false, if it contains no data.

Example

The following example checks if any part of the summary_info element has been filled in.

```
if (po.getElement("summary_info").hasData() == false) {
  System.out.println("The summary info has not been filled in
          for PO " + po.getData("po_number"))
}
```

isField method

This method checks whether this element is a field or group. This procedure is useful when you want to use code that can manipulate different business object types, for example.

Declaration

public abstract boolean isField()

Return value

Boolean true if this element is a field element; boolean false if this element is a group element.

Example

The following example supports two different business object types: one type has ship_to as a group with an address field within it, while the other type has ship_to as a field holding address data. The following piece of code is reusable for both cases.

```
public String getShippingAddress(Element ship_to) {

  if (ship_to.isField()) {
      return ship_to.getData();
  } else {
         return ship_to.getData("address");
  }
}
```

isValid method

This method determines the validity of an element based on its business object type.

- If the element you're checking is a group, isValid determines whether all mandatory fields it contains have data; if a subordinate optional group contains data, it also checks whether all mandatory fields in the optional group have data. If a subordinate optional group doesn't have data, it's ignored. For subordinate element sequences, each element is checked individually for validity. For mixed content, the mixed content must not be null or " ".
- If the element is a field, isValid checks whether it contains data or not.
- If the element is an attribute, isValid checks that it is not null or " "; if the attribute declaration has a list of legal values and the current value is in that list; the attribute is #FIXED and its current value is null or " " or the default value.

isValid does not consider whether the element you called isValid on was optional or mandatory when it determines validity. For fields, it only checks whether the field has data or not; for groups, isValid checks the elements it contains for validity based on whether they are optional or mandatory. It is often useful to invoke isValid method in conjunction with the hasData method, whenever operating on any optional element.

Note: An empty string ("") as a data value is equivalent to null. A string containing one or more spaces (" ") is not null, and is data.

Declaration

public abstract boolean isValid()

Return value

Boolean true if this element is a field containing data or is a group whose subordinate elements are valid (or are optional and contain no data); boolean false if this element is a field with null data or is a group with a mandatory subordinate field that has null data.

Example

The following example checks the validity of the top-level element of a business object. It prints an error message if the po business object is not valid:

```
if (po.isValid() == false) {
  System.out.println("The Purchase_Order has unfilled
       mandatory fields: " + po.toString(true));
}
```

See Also

hasData

removeAttr method

This method removes the specified attribute.

Declaration

void removeAttr(String *attribute_name*) throws InvalidQueryException

Parameter

attribute_name is a string indicating the name of the attribute.

Exceptions

InvalidQueryException indicates that the attribute name you supplied is invalid. For example, you can get this error if you typed a wrong name.

Example

This example removes the attribute my_attribute.

```
po.removeAttr("my-attribute");
```

See Also

getAttr, setAttr, InvalidQueryException, InvalidDataException

removeAttrs method

This method removes all attributes.

Declaration

void removeAttr()

Example

This example removes all attributes.

```
po.removeAttrs();
```

See Also

getAttr, setAttr, removeAttr

removeElement method

This method removes the specified element. If the element doesn't exist, nothing is removed. If the tag_path refers to an element in a sequence, this is equivalent to calling removeElementAt(index).

Declaration

void removeElement(String *tag_path*) throws InvalidQueryException, IndexOutOfBoundsException

Parameter

tag_path is a string indicating the location of the element or group in the element tree.

Exceptions

InvalidQueryException indicates the tag path you supplied is invalid. If an array index is specified for a parent that is not a sequence, this exception is thrown.

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

Example

This example removes the attribute my_attribute.

```
po.removeElement("my_element");
```

See Also

removeElementAt, InvalidQueryException, InvalidDataException

setAttr method

This method sets the value of the specified attribute. To remove the attribute, set the attribute value to null.

Declaration

setAttr(String *attribute_name*, String *attribute_value*) throws InvalidQueryException, InvalidDataException

Parameter

attribute_name is a string indicating the name of the attribute.

attribute_value is a string indicating the value to set for this attribute.

Exceptions

InvalidQueryException indicates that the attribute name you supplied is invalid. For example, you can get this error if you typed a wrong name. This is also thrown if you specify an array index for a non-sequence, or if you didn't specify an array index for a sequence.

InvalidDataException indicates that the attribute declaration has a list of valid values and the value you supplied is invalid.

Example

This example returns the value of the attribute my_attribute. *value* holds the returned string.

```
String value;
value = po.getAttr("my-attribute");
```

See Also

getAttr, InvalidQueryException, InvalidDataException

setData method

This method sets the data contained in this field element or in the field element identified by the tag path string. If the tag path is valid, but the element doesn't exist, the element is created. If you set the data for an attribute to be null, this removes the attribute.

Note: An empty string ("") as a data value is equivalent to null. A string containing one or more spaces (" ") is not null, and is data.

Declaration

public abstract void setData(String *data*) throws ElementTypeException

public abstract void setData(String *tag_path*, String *data*) throws InvalidQueryException, ElementTypeException, InvalidDataException, IndexOutOfBoundsException

Parameters

data String representation of the data to be contained in this field. A null or empty string clears any data currently contained in this element. To specify an attribute, use the @ notation. For example, foo@bar would be the attribute bar of the element foo.

tag_path is a string indicating the location of the element in the element tree.

Exceptions

ElementTypeException indicates that you called this method on a group element and the group doesn't support mixed content.

InvalidQueryException indicates that the tag path you supplied is invalid. For example, you can get this error if you typed a wrong name. This is also thrown if you specify an invalid attribute. This is thrown if you called this method on a group and that group does not support mixed content.

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

InvalidDataException indicates that you tried to set data for an attribute that doesn't conform to the attribute declaration list of valid values.

Examples

This example sets the po_number field to the string 24567:

```
po_num_el.getElement("po_number").setData ("24567");
```

This example does exactly the same thing:

```
String po_num;
po_num = "24567";
po_num_el.setData ("po_number", po_num);
```

See Also

ElementTypeException, InvalidQueryException, IndexOutOfBoundsException

toString method

This method returns a string describing the validity and content of this element and any elements subordinate to it that contain data. Mandatory subordinate elements always appear in the description; optional subordinate elements without any data do not appear. This helps you see which elements must have data for this element to be valid. You can use the println procedure to display the value returned by toString.

Declaration

public abstract String toString(boolean *include_data*)

Parameters

include_data is a boolean. If it is true, toString includes field data values in the returned string; if false, it doesn't include the data values, which makes the description shorter.

Return value

String describing the element.

Example

This example prints the description string, including the data values:

```
System.out.println(po.toString(true));
```

The output might look like this (note that the top-level Purchase_Order element isn't valid because the mandatory field po_date has no value):

```
<Purchase_Order valid="false">
<po_number valid="true">123</po_number>
<po_date valid="false"></po_date>
<supplier_id valid="true">99999</supplier_id>
<po_line valid="true">
<item_code valid="true">2222</item_code>
<qty valid="true">55</qty>
<expected_ship_date valid="true">9.9.99</expected_ship_date>
<summary_info valid="true">
<comments valid="true">first line item</comments>
</summary_info>
</po_line>
</Purchase_Order>
```

ElementSequence interface

An ElementSequence is a collection of consecutive Elements of the same element type. An ElementSequence might contain a set of either group elements or field elements. This interface provides a way to manage the number of these consecutive, repeatable Elements.

Every ElementSequence is associated with a specific location in the logical Element tree of a BO instance. The legal locations of ElementSequences are determined by element declarations in the BO type definition. If an element type declaration has specified that a particular child element type might occur zero-or-more or one-or-more times in its content model, then an ElementSequence might be used to add and remove child elements of that type.

Note that if a sequence of elements is said to be optional, it doesn't mean that members of the sequence are not required to contain data. It means that the sequence might legally be of length zero. If an Element is added to any sequence, optional or mandatory, the Element must be valid for its parent to be valid.

getElementAt method

This method gets a reference to the group or field at the specified position in an element sequence.

Declaration

public abstract Element getElementAt(int *index*) throws IndexOutOfBoundsException

Parameters

index is an integer and is the index of the Element to return.

Return value

Element, the element at the specified position.

Exceptions

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

Example

This example gets the element at the position i.

```
int i = 0;
Element single_line;
single_line = po.getElementSequence("po_line").getElementAt(i);
```

See Also

getElementSequence, IndexOutOfBoundsException

hasData method

This method checks whether the element sequence contains data.

Note: An empty string ("") as a data value is equivalent to null. A string containing one or more spaces (" ") is not null, and is data.

Declaration

public abstract boolean hasData()

Return value

Boolean true if any element in this sequence contains data; boolean false if no element contains no data.

Example

The following example checks if there are any design_drawing elements in the po business object.

```
if (po.getElementSequence("design_drawing").hasData() == false)
{
  System.out.println("There's no design drawing associated with
          this PO " + po.getData("po_number"));
}
```

LENGTH METHOD

This method returns the number of elements in this element sequence. This is useful for setting boundary values to loop through all the elements in an element sequence.

DECLARATION

public abstract int length()

RETURN VALUE

length returns an integer, the number of elements in the sequence.

EXAMPLE

This example loops through all the po_line elements and prints the data in each po_line in string form:

```
int lines, nLines, i;
ElementSequence lines;
lines = po.getElementSequence("po_lines");
nLines = lines.length();
for (i=0; nLines - 1; i++) {
  System.out.println(lines.getElementAt(i).toString(true));
}
```

SEE ALSO

getElementSequence

newElement method

This method adds a new element to the end of this sequence and returns a reference to the newly created element. The new length of the sequence will be length and the index of the new element will be length - 1.

Declaration

Element newElement()

Return value

Element, the new empty element.

Example

This example creates a new po_line element at the end of the sequence and copies into it the value of the element before it in the sequence:

```
lines = po.getElementSequence("po_line");
nlines = lines.length();
last_line_index = nlines - 1;
new_line = lines.newElement();
if (last_line_index >= 0) {
  new_line.copyIn(lines.getElementAt(last_line_index));
}
```

See Also

newElementAt

newElementAt method

This method inserts a new element at the specified position in the sequence and returns a reference to the newly created element. Adds 1 to the index of the element currently in that position (if any) and any following elements, so they are shifted to the right.

Valid index values are 0 to length. If the index is length, an element is added to the end of the sequence.

Declaration

public abstract Element newElementAt(int *index*) throws IndexOutOfBoundsException

Parameters

index specifies a position in the sequence.

Return value

Element the new empty element.

Exceptions

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

Example

This example adds a new po_line element at the beginning of a sequence:

```
line = po.getElementSequence("po_line").newElementAt(0);
```

See Also

newElement

removeAll method

This method removes all elements in this sequence. Any data contained in any of the elements is deleted. The length of this element sequence becomes 0.

Declaration

public abstract void removeAll()

Example

The following example removes all the po_line elements from the po business object:

```
po.getElementSequence("po_line").removeAll;
```

See Also

removeElementAt

removeElementAt method

This method removes the element at the specified position in this sequence and its data. The indices of elements at greater index values are reduced by 1 (they are shifted to the left). After the call successfully completes, the length of the sequence is reduced by 1.

Valid index values are 0 to length() - 1.

Declaration

public abstract void removeElementAt(int *index*) throws IndexOutOfBoundsException

Parameters

index specifies an element in the sequence (remember indices start at zero [0]).

Exceptions

IndexOutOfBoundsException indicates that, when specifying an element in a sequence, you provided an invalid index value, such as a number greater than length - 1 for that element sequence.

Example

The following example loops through all po_line elements and removes all the invalid ones. Note that removeElementAt will move the remaining elements one index down in the sequence. Perform this operation starting at the end of the sequence so your index value is always valid.

```
Element line;
ElementSequence lines;

lines = po.getElementSequence("po_line");
for (i = lines.length() - 1; i >= 0; i--) {
  line = lines.getElementAt(i);
  if (!line.isValid()) {
      lines.removeElementAt(i);
  }
}
```

See Also

removeAll, removeElement

Exceptions

ElementTypeException

This exception is used to indicate one of two types of errors that can occur when using Element methods:

- An element type mismatch in a call to one of the copyIn methods. copyIn only supports copying between elements of the same type from the same Element Definition Set.
- Use of a field Element method on a group element, or vice versa. Certain methods are defined to only operate on field elements or group elements. Examples are getData and setData which might only operate on field elements.

Thrown By:

copyIn, getData, getDataType, setData

InvalidQueryException

This exception is thrown when a tag path used in an Element method is not valid according to the element's content definition. Cases where this exception will be thrown include:

- Using a non-empty tag path when calling a method on a field element. Field elements never contain child elements.
- Mistyping a subordinate element's name in a tag path. Note that element names in tag paths are case-sensitive.
- Using an index with an element in the tag path that is not repeatable.
- Failing to use an index with a repeatable element when accessing an element or element sequence within that repeatable element.
- Failing to use an index with the last element in a tag path when calling getElement to access a member of a sequence.
- Using an index with the last element in a tag path when calling getElementSequence to access an element sequence.

Note that in cases where an index value in a tag path is out of bounds, an IndexOutOfBoundsException will be thrown.

Thrown By:

copyIn, getData, getElement, getElementSequence, setData

IndexoutOfBoundsException

This exception is thrown when an index value in a tag path is out of bounds. Cases where this exception will be thrown include:

- Using a tag path index that is greater than length -1 of the ElementSequence.
- Using a tag path index that is less than 0.

Thrown By:

copyIn, getData, getElement, getElementSequence, setData, getElementAt, newElementAt, removeElementAt

CHAPTER

10

ADDING CUSTOM CODE TO ADAPTERS

Read this chapter for information about adding custom code to adapter implementations. For detailed information about the Partner Agreement Manager Adapter API, see *Using Business Object API methods* on page 191.

This chapter includes these sections:

About adding custom code

When you generate code for an adapter, Partner Agreement Manager provides stubbed code for the operations you specify. The generated code contains "To Do" sections that identify the areas of the code that you need to implement.

The next step is to open the generated code in a text editor or the integrated development environment (IDE) of your choice and add the required code. The code you add depends on the operations your adapter performs and the interfaces it connects to.

Before you add code to a generated adapter, you might want to consider using helpers, life cycle methods, execution methods, event production methods, and libraries. The sections that follow describe considerations for each area.

Tip: Generating code is a write-only operation. Changes you make in generated code are not maintained if you generate new code. Therefore, if you change an adapter type and generate new code, or generate new code for an adapter implementation, any changes you have made to the generated code are lost unless you use a different file name for the generated code. (The Adapter Designer prompts you before overwriting files.) To minimize the amount of cut and paste required, you should make a practice of making calls outside of the generated code to a separate module(s) that contain the actual business logic code. This way, once a business logic module is debugged and ready for production, a change to the design of the adapter type itself will not introduce new bugs to the business logic.

Using helpers

As you develop adapter implementations, it's a good idea to maintain business logic for all but the most trivial adapters in a separate helper file. If you find later that you need to revise your adapter type—to add a new operation, for example—you will also need to generate a new implementation, and then merge the logic contained in your old file with the newly generated file.

You might find it more practical to put all adapter entry points in a helper file that contains all the substantive adapter logic. This allows you to regenerate the adapter implementation Java source code file freely and often. Then, you copy and paste snippets of code from the helper file into five Java adapter implementation sections: Variables, Constructors, Life cycle methods, Execution methods, and Event Production methods.

The following are examples of method calls from the sample InventoryAdapter into its InventoryAdapterHelper.

Declaration of the helper variable and instantiation in the constructor:

```
public class InventoryAdapter extends Adapter{

private InventoryAdapterHelper helper;

public InventoryAdapter() {
  super();
  this.helper = new InventoryAdapterHelper();
}
```

Life cycle method—in startup(AdapterContext context):

```
helper.startup(jdbc_driver, database_url, user, password,
         reorder_quantity);
```

Execution method—in executeGetInventoryInformation(OperationContext context)

```
helper.getInventoryInformation(item_number, output_bo,
         status_bo);
```

Event Production method—in checkForEvents(EventContext context)

```
helper.checkForEvents(context);
```

Following are the corresponding method signatures in the helper class:

Life cycle method

```
void startup(String jdbc_driver, String database_url,
         String user, String password, int reorder_quantity)
```

Execution method

```
void getInventoryInformation(String item_str,
         BusinessObject output_bo,BusinessObject status_bo)
```

Event Production method

```
void checkForEvents(EventContext context)
```

The helper method signatures can look very similar to those in the generated adapter. Helper life-cycle and execution methods typically take arguments unpacked from the AdapterContext or OperationContext, respectively. If more convenient, they can take the entire context object. Helper event production methods can just receive the original EventContext object.

Using life cycle methods

Life cycle methods are called when the adapter is started for the first time. They include the startup and shutdown methods. When you generate an adapter implementation, you can set the implementation to include stubs for sections that override standard life cycle behavior.

Startup validation

Use the startup method to perform startup validation to check property values at this point (if applicable or possible), before the adapter begins to execute operations. This allows you to report configuration errors while the user starts the adapter or Adapter Server. Startup is also the place to connect to business applications such as databases. If the attempt to connect to the business system fails, throw an EndSystemNotAvailableException exception. This will be caught be the Adapter Server. If you set error handling in the adapter instance, the Adapter Server will suspend the failed adapter and attempt to restart it, in case the connection error was momentary. For more on setting error handling in adapter instances, see the *Partner Agreement Manager Administrator's Guide*.

String properties

When checking string property values, remember the difference between the null string and the empty string ("").

You might want to treat the empty string as equivalent to null, the absence of a value. Code constructs like the following trap both cases:

```
if (prop != null && prop.length() != 0) {
  // do something
}  else {
  // do something else
}
```

Integer properties

Be careful about optional integer properties. If left unbound in an adapter instance, the default loadAdapterProperties method in generated Java implementations will not attempt to set a value for the generated integer variable.

However, the Java language's default value for uninitialized scalar integers is zero (0). You might want to change the generated integer variable's default value to something more appropriate—for example, something that would not be an actual property value.

Before:

```
protected int optional_integer;
```

After:

```
protected int optional_integer = 17;
```

Because the Adapter Designer requires you to choose a default value for both mandatory and optional boolean properties, you do not encounter this issue with boolean properties.

Exception handling

It's a good idea to throw an EndSystemNotAvailableException if your adapter needs to connect with an outside system but fails to do so. This will trigger the Adapter Server to suspend this adapter and attempt to retry it. The Adapter Server will attempt retries until it reaches the maximum number of recovery attempts, as set in the adapter implementation properties. See the *Partner Agreement Manager Administrator's Guide* for more information on setting retry parameters.

Using execution methods

Execution methods are called to execute a specific operation and can be called by several threads. You can manipulate business objects generically using the classes in the com.extricity.document.api package. See *Using Business Object API methods* on page 191 for more information.

Exception handling

You can handle exceptions in one of two ways: in the adapter or in the private process logic. If you throw exceptions in the adapter, you run the risk of aborting the process when an error occurs. If this exception is encountered, the process aborts: you will see the exception in the audit log and stack trace. If you're sure you know how to handle the exception, you can catch application exceptions and pass them back in the status business object. This will give a process designer the opportunity to handle the error condition in the private process. Make sure that the description for your operation clearly states what exceptions can be reported in the status business object. If you are not entirely sure you know how to handle an exception, let the Adapter Server handle it.

From the private process, you can branch from different possible success, warning, and failure outcomes. For example, different branches can notify an individual of the error, continue processing, or abort the process using the Termination action. For different errors, you can notify different individuals on different error paths. For example, the private process can notify the database administrator if the purchasing database is down, or the order administrator if an order has not yet been approved.

The key is to give process designers the flexibility to do whatever they need to do.

That said, you might want to throw ISException from an execution method during the early stage of an implementation. As an adapter developer, you will likely be in an iterative development cycle—starting many processes and restarting the Adapter Server and adapters many times. Therefore, you might want the process to abort at the first sign of an adapter error. As your implementation nears the production cutover, however, errors can be relayed back to the private process for notification of business users.

Do not catch any exceptions that you do not know how to handle. Whatever exceptions you catch should be named.

This construct is preferable:

```
} catch (IOException e) {
  // report IO problem in status BO
}
```

To this one:

```
} catch (Exception e) {
  // report unknown problem in status BO
}
```

The second construct might hide some condition that should genuinely stop process execution (for example, out of memory, disk full).

Long-running operations

The requestRetry method (defined in the OperationContext class) allows you to programmatically specify a long-running operation. It defines how long you want to wait before the Adapter Server asks your execution method again for its result.

```
context.requestRetry(int seconds):
```

You can manage this flexibility through properties or operation inputs. In deciding which approach to use, consider if a private process designer is likely to need this type of control. The more system-oriented or low-level the timing specification, the more likely it is that the configuration should be managed in the adapter properties.

Here are some examples of ways you can manage the flexibility.

Options with no user input:

- appropriate constant value for requestRetry
- exponential backoff algorithm (tries again in 1 minute the first time, tries again in 2 minutes the second time, then 4 minutes, then 8 minutes, and so on)

Configured per adapter instance (applies across all operations):

- single property in one time unit (for example, minutes)
- several properties up to four time units; you can have a property each for seconds, minutes, hours, and days—a property with static list of likely time period values

Configured per operation execution:

- time specified in input BO field
- time specified in input variant

For example, the example Purchasing adapter contains a method that looks to determine if it should call requestRetry based on the status of orders. If an order has been approved, the method returns a value of true.

If, however, the order has not been approved, the value is false, and the adapter must call requestRetry to check again for approval after the specified period of time.

```
if (!this.helper.waitForPurchaseOrderApproval(po_number,
          output_bo,status_bo)) {
  context.requestRetry(this.wait_polling_interval);
}
```

Using event production methods

Event production methods are called to check for new Adapter Server events. They can be either stateful or stateless (see the *Partner Agreement Manager Administrator's Guide*).

Exception handling

Here you should throw an ISException if you encounter an error. No events will initiate Partner Agreement Manager processes, but this is the best way to communicate problem conditions to the Adapter Server window. You can also throw an EndSystemNotAvailableException if you detect a lost connection during a checkForEvents.

You might also define an event that gets returned to Partner Agreement Manager after catching an exception. Then a process can be run to send the exception message to an administrator.

Developing adapter class libraries

A class library provides Java methods for a business system, a technical interface, or an industry standard protocol. Class libraries can have static methods—methods that don't need to be called on a Java object instance—and can have no state.

About compiling, testing, and debugging adapters

Following are some tips for testing adapters and implementing exception handling for easier troubleshooting.

Testing adapters

Because the Java class files are unloaded when an adapter is stopped, recompilation changes take effect when the adapter is restarted, rather than requiring you to restart the Adapter Server.

Some issues to bear in mind when testing adapters are:

- main method: Your adapters, adapter helpers, and library classes should have a main method for testing from the command prompt. Almost every class can perform useful sanity checks in the main method. If your adapter doesn't work by itself, it most likely won't work in a process. (See `JDBCLibrary.java` and `InventoryAdapterHelper.java` for examples.)
- Use your adapter in a single-node public process with a simple private process for unit testing. You should catch as many errors as possible before integration testing with your partners.
- If you use the Jikes compiler, add the command line argument `-g` when testing to embed line numbers in the class file.
- If you use the Microsoft JVC compiler, add the command-line arguments `/g` or `/g:l` when testing to embed line numbers in the class file.

Implementing exception handling

When you throw an ISException to wrap an underlying exception, use the ISException constructors that allow you to pass in the nested exception:

```
public ISException(Throwable e)
public ISException(String message, Throwable e)
```

Seeing the nested exception will make it easier for troubleshooters to find and fix the source of the problem.

If you encounter a run-time exception and still need to perform some cleanup operations, catch any cascading exception so you can yield the earlier exception. The earlier exception will generally be more closely related to the problem. See the Inventory Example Helper and JDBC Library for an example.

APPENDIX

A

Java implementation example

Read this appendix for a full description of the sample Inventory adapter that is included in the Adapter Development Environment.

This appendix includes these sections:

About the Java implementation example

This section presents a sample Inventory database adapter type for the Java implementation, including the full source code. There are three implementations of the inventory database as follows:

- Generated Inventory Adapter — the generated implementation works, but doesn't do much. It's a minimal, stub implementation.
- Testing Inventory Adapter — the testing implementation illustrates a very simple in-memory database that does not require an actual JDBC connection.
- Inventory Adapter — the real implementation uses a JDBC connection and illustrates a functional adapter. It also illustrates the use of a helper class that isolates the JDBC code from the Adapter Designer code.

Note: The source code for this example is on the Partner Agreement Manager CD and is installed in your partner directory: `<partner root>\com\extricity\adapters\Example\inv` or `\jdbc` or `\purch`

The Example Inventory Adapter type

This example Partner Agreement Manager adapter for a simple inventory system demonstrates Get and Post operations, as well as event polling. This adapter needs to be accompanied by the Example_Inventory_Item.1 BO. This adapter also makes use of the Operation_Status.1 BO.

Note: Before running any operations, be sure to set up your simple Inventory system by running the accompanying SQL schema script to create the Inventory database table(s).

Properties	jdbc_driver (Java class name of the database driver)
	database_url (database URL of the form jdbc:subprotocol:subname)
	user (database user name)
	password (database user's password)
	reorder_quantity (on-hand quantity below which items need to be reordered)

Operations

GET: Get inventory information

Description: Retrieves information about an item from the Inventory database and stores it into the output Example_Inventory_Item.1 BO.

Returns Operation_Status.result of “success” if item is found and information is successfully queried.

Returns Operation_Status.result of “failure” if item_number input field is not an integer. Also returns “failure” if a database error occurs. Also returns “failure” if no item with the given number exists in the database.

Input: item_number — The queried item number variant

Output: Extricity.3.Example_Inventory_Item.1 BO

Status: Extricity.3.Operation_Status.1 BO

POST: Create inventory item

Description: Creates a new item in the Inventory database with the data in the input Example_Inventory_Item.1 Business Object.

Returns Operation_Status.result of “success” if item is successfully created.

Returns Operation_Status.result of “failure” if item_number or quantity_on_hand input BO fields are not integers. Also returns “failure” if a database error occurs. Also returns “failure” if an item with the given item_number already exists in the database.

Input: Extricity.3.Example_Inventory_Item.1 BO

Status: Extricity.3.Operation_Status.1 BO

POST: Update item quantity

Description: Updates Inventory database with a new on-hand quantity using the data in the input Example_Inventory_Item.1 BO. Ignores the description and warehouse_id BO fields.

Returns Operation_Status.result of “success” if item quantity is successfully updated.

Returns Operation_Status.result of “failure” if item_number or quantity_on_hand input BO fields are not integers. Also returns “failure” if a database error occurs. Also returns “failure” if no item with the given number exists in the database.

Input: Extricity.3.Example_Inventory_Item.1 BO

Status: Extricity.3.Operation_Status.1 BO

Events

EVENT: Example inventory item below reorder quantity

This event indicates that inventory levels have fallen below the reorder level.

Generated adapter implementation

The Generated Inventory Adapter Java Implementation is the Java source code file exactly as it was generated from the Adapter Designer, except for the initial copyright header and the *FileVersion* constant.

This is a compilable, functioning adapter, but only in the barest sense. It will start up and shut down properly. It will also poll for events, though no events will ever be returned. However, the Get operations will not work properly and return an invalid output BO because mandatory fields in the *Example_Inventory_Item.1* Business Object will not be set. Moreover, both Get and Post operations will return an invalid status BO because the mandatory "result" field of the *Operation_Status.1* Business Object will not be set. Properties will be ignored.

```
//---------------------------------------------------------------------------
// Source code auto-generated by:
//
// Adapter Designer
//
// Generated on: Sat Mar 04 17:19:34 PST 2001
//---------------------------------------------------------------------------


package com.extricity.adapters.example.inv;


import com.extricity.adapter.api.*;
import com.extricity.document.api.*;

/*
 * The Generated Inventory Adapter Java Implementation
 *
 * This is the Java source code file exactly as it was generated from the
 * Adapter Designer except for the initial copyright header, this comment,
 * and the FileVersion constant.
 *
 * This is a compilable, functioning adapter, but only in the barest sense.
 * It will start up and shut down properly.  It will also poll for events,
 * though no events will ever be returned.  However, the Get operations will
 * not work properly and return an invalid output BO because mandatory fields
 * in the Example_Inventory_Item.1 Business Object will not be set.
 * Moreover, both Get and Post operations will return an invalid status BO
 * because the mandatory "result" field of the Operation_Status.1 Business
 * Object will not be set.  Properties will be ignored.
 */
/**
 * This is an example Partner Agreement Manager adapter for a simple Inventory system.
    It demonstrates Get and Post operations as well as event polling.
 *
 * This adapter should be accompanied by the Example_Inventory_Item.1 BO.
 * This adapter also makes use of the Operation_Status.1 BO.  Before running
 * any operations, be sure to set up your simple Inventory system by running
 * the accompanying SQL schema script to create the Inventory database
 * table(s).
 *
 *
```

```
public class GeneratedInventoryAdapter extends Adapter
{
   static final String FileVersion = "$Revision: 5 $";

   //------------------------------------------------------------------------
   // Constants - Protected - Internal
   //------------------------------------------------------------------------

   protected static final String UNKNOWN_OPERATION_TYPE_ID =
      "Trying to execute an Adapter operation with an unknown operation type";
   protected static final String UNKNOWN_OPERATION_ID =
      "Trying to execute an unknown Adapter operation";


   //------------------------------------------------------------------------
   // Constants - Protected - Adapter Properties
   //------------------------------------------------------------------------

   protected static final String PROP_REORDER_QUANTITY = "reorder_quantity";
   protected static final String PROP_JDBC_DRIVER = "jdbc_driver";
   protected static final String PROP_PASSWORD = "password";
   protected static final String PROP_DATABASE_URL = "database_url";
   protected static final String PROP_USER = "user";


   //------------------------------------------------------------------------
   // Constants - Protected - Adapter Events
   //------------------------------------------------------------------------

   protected static final String EVENT_EXAMPLE_INVENTORY_ITEM_BELOW_REORDER_QUANTITY =
     "Example inventory item below reorder quantity";


   //------------------------------------------------------------------------
   // Constants - Private - Post Operations
   //------------------------------------------------------------------------

   // "Create inventory item" constants
   protected static final String CREATE_INVENTORY_ITEM = "Create inventory item";

   // "Update item quantity" constants
   protected static final String UPDATE_ITEM_QUANTITY = "Update item quantity";


   //------------------------------------------------------------------------
   // Constants - Private - Get Operations
   //------------------------------------------------------------------------

   // "Get inventory information" constants
   protected static final String GET_INVENTORY_INFORMATION = "Get inventory
     information";
   protected static final String GET_INVENTORY_INFORMATION_INPUT_ITEM_NUMBER =
     "item_number";


   //------------------------------------------------------------------------
   // Variables - Protected - Adapter Properties
   //------------------------------------------------------------------------

   /**
    * The on hand quantity below which items should be reordered.
    */
   protected int reorder_quantity;

   /**
    * The Java class name of the database driver.
    */
```

```
protected String jdbc_driver;

/**
 * The user's password.
 */
protected String password;

/**
 * A database URL of the form jdbc:subprotocol:subname .
 */
protected String database_url;

/**
 * The database user on whose behalf the Connection is being made.
 */
protected String user;


//------------------------------------------------------------------------
// Constructors
//------------------------------------------------------------------------

/**
 * Default constructor.
 */
public GeneratedInventoryAdapter() {
   super();
}


//------------------------------------------------------------------------
// Methods - Lifecycle
//------------------------------------------------------------------------

/**
 * Called when the adapter is started for the first time.
 *
 * @param context Holds Adapter specific properties, may be empty.
 *
 * @exception ISException Thrown if there is a problem starting the
 * Adapter.
 */
public void startup(AdapterContext context)
   throws ISException {

   // load the properties from the adapter
   loadAdapterProperties(context);

   // add startup code here
}


/**
 * Called when the adapter is stopped. Allows the adapter to perform any
 * necessary cleanup work.
 *
 * @param context Holds Adapter specific properties, may be empty.
 *
 * @exception ISException Thrown if there is a problem starting the
 * Adapter.
 */
public void shutdown(AdapterContext context)
   throws ISException {

   // add shutdown code here
}
```

```
//-------------------------------------------------------------------------
// Methods - Execution
//-------------------------------------------------------------------------

/**
 * Called to execute a specific operation. This method must be overridden
 * by all Adapter subclasses.
 *
 * This method can be called by multiple threads.
 *
 * If the adapter subclass supports transactions, the transaction will be
 * either commited after this method returns or rolled back if an exception
 * is thrown. If the Adapter subclass does not support transactions, it
 * must either commit or rollback all stateful actions performed during
 * execution of this method.
 *
 * @param context Runtime context for the operation. Container for
 * operation inputs and outputs (output values must be added by the Adapter
 * subclass implementation).
 *
 * @exception ISException Thrown if there is an error during execution.
 */
public void execute(OperationContext context)
   throws ISException {
   int operation_type_id;
   String operation_id;

   operation_type_id = context.getOperationTypeID();
   operation_id = context.getOperationID();

   switch (operation_type_id) {
   case OperationContext.GET:
      if (operation_id.equals(GET_INVENTORY_INFORMATION)) {
         executeGetInventoryInformation(context);
      } else {
         throw new ISException(UNKNOWN_OPERATION_ID + ": " + operation_id);
      }
      break;

   case OperationContext.POST:
      if (operation_id.equals(CREATE_INVENTORY_ITEM)) {
         executeCreateInventoryItem(context);
      } else if (operation_id.equals(UPDATE_ITEM_QUANTITY)) {
         executeUpdateItemQuantity(context);
      } else {
         throw new ISException(UNKNOWN_OPERATION_ID + ": " + operation_id);
      }
      break;

   case OperationContext.ADVANCED:
      break;

   default:
      throw new ISException(UNKNOWN_OPERATION_TYPE_ID + ": " + operation_type_id);
   }
}


/**
 * Retrieves information about an item from the Inventory database and
 * stores it into the output Example_Inventory_Item.1 BO.
 *
 * Returns Operation_Status.result of "success" if item is found and
 * information is successfully queried.
 *
 * Returns Operation_Status.result of "failure" if item_number input field
 * is not an integer.  Also returns "failure" if a database error occurs.
 * Also returns "failure" if no item with the given number exists in the
 * database.
 */
protected void executeGetInventoryInformation(OperationContext context)
   throws ISException {
   String item_number;
   Element status_bo;
   Element output_bo;

   // load the inputs from the OperationContext
   item_number =
  context.getInputVariant(GET_INVENTORY_INFORMATION_INPUT_ITEM_NUMBER);

   // create the output Business Objects
   output_bo = context.createOutputBOElement();
   status_bo = context.createStatusBOElement();

   // add execute code here
}


/**
 * Updates Inventory database with a new on-hand quantity using the data in
 * the input Example_Inventory_Item.1 Business Object.  Ignores the
 * description and warehouse_id BO fields.
 *
 * Returns Operation_Status.result of "success" if item quantity is
 * successfully updated.
 *
 * Returns Operation_Status.result of "failure" if item_number or
 * quantity_on_hand input BO fields are not integers.  Also returns
 * "failure" if a database error occurs.  Also returns "failure" if no item
 * with the given number exists in the database.
 */
protected void executeUpdateItemQuantity(OperationContext context)
   throws ISException {
   Element input_bo;
   Element status_bo;

   // load the inputs from the OperationContext
   input_bo = context.getInputBOElement();

   // create the output Business Objects
   status_bo = context.createStatusBOElement();

   // add execute code here
}


/**
 * Creates a new item in the Inventory database with the data in the input
 * Example_Inventory_Item.1 Business Object.
 *
 * Returns Operation_Status.result of "success" if item is successfully
 * created.
 *
 * Returns Operation_Status.result of "failure" if item_number or
 * quantity_on_hand input BO fields are not integers.  Also returns
 * "failure" if a database error occurs.  Also returns "failure" if an item
 * with the given item_number already exists in the database.
 */
protected void executeCreateInventoryItem(OperationContext context)
   throws ISException {
   Element input_bo;
   Element status_bo;
```

```
      // load the inputs from the OperationContext
      input_bo = context.getInputBOElement();

      // create the output Business Objects
      status_bo = context.createStatusBOElement();

      // add execute code here
   }


   //---------------------------------------------------------------------------
   // Methods - Runtime - Event Production
   //---------------------------------------------------------------------------

   /**
    * Called to check for new IS events. Produced events must be added to the
    * EventContext.
    *
    * @param context Context for the check call, functions as container for
    * produced events.
    *
    * @exception ISException Thrown if there is an error checking for the
    * event.
    */
   public final void checkForEvents(EventContext context)
      throws ISException {

      // add event production code here
   }


   //---------------------------------------------------------------------------
   // Methods - Private
   //---------------------------------------------------------------------------

   /**
    * Loads values for declared properties from the AdapterContext.
    */
   private void loadAdapterProperties(AdapterContext context)
      throws ISException {

      if (context == null) {
         return;
      }

      jdbc_driver = context.getPropertyAsString(PROP_JDBC_DRIVER);
      password = context.getPropertyAsString(PROP_PASSWORD);
      database_url = context.getPropertyAsString(PROP_DATABASE_URL);
      user = context.getPropertyAsString(PROP_USER);

      if (context.isPropertyBound(PROP_REORDER_QUANTITY)) {
         reorder_quantity = context.getPropertyAsInt(PROP_REORDER_QUANTITY);
      }
   }
}
```

Testing inventory adapter implementation

This section describes the Testing Inventory Adapter Java Implementation. This adapter implementation is useful for testing the interface of the example Inventory Adapter Type in Partner Agreement Manager private processes without having a real database connection.

It actually implements an in-memory inventory database of a single item. Later you can see how this stubbed implementation is quite similar to the real JDBC implementation in the InventoryAdapter and InventoryAdapterHelper classes.

Here is how the simulated behavior differs from the real implementation:

This item	Does this
startup	Only caches reorder quantity property; does not open any database connection.
shutdown	Nothing; does not close any database connection.
Get inventory information (Get)	Gets the information if the input item number matches the only item in memory.
Create inventory item (Post)	Creates the item if it does not exist in the database (in other words, if the new item number is different than the single existing number); replaces the old item in memory.
Update item quantity (Post)	Updates the item quantity only if the item number matches the only item in the database.
Events	Sees if the single item is below reorder quantity, but can generate the one event per checkForEvents call.

- Because the item is stored only in memory, the single item will not persist if you shut down the Adapter Server, unlike the real JDBC implementation.
- All properties except for the reorder_quantity (in other words, the JDBC properties) will be ignored.

NOTE: The modifications from the generated Java implementation appear in **bold.** You can also look for comments in this form:

```
// XXX added for TestingInventoryAdapter
//---------------------------------------------------------------------------
// Source code auto-generated by:
//
// Adapter Designer
//
// Generated on: Sat Mar 04 17:19:40 PST 2001
//---------------------------------------------------------------------------

package com.extricity.adapters.example.inv;


import com.extricity.adapter.api.*;

import com.extricity.document.api.*;


/*
 * The Testing Inventory Adapter Java Implementation.
 *
 * This adapter implementation is useful for testing the interface of the
 * Example Inventory Adapter Type in Partner Agreement Manager private processes
 * without having a real database connection.
 *
 * It actually implements an in-memory inventory database of a single item.
 * Later you can see how this stubbed implementation is quite similar to
 * the real JDBC implementation in the InventoryAdapter and
 * InventoryAdapterHelper classes.
 *
 * Here is how the simulated behavior will differ from the real
 * implementation:
 *
 * startup: only caches reorder quantity property; does not open any database
 *    connection
 *
 * shutdown: does nothing; does not close any database connection
 *
 * Get inventory information (Get): gets the info if the input item number
 *    matches the only item in memory
 *
 * Create inventory item (Post): creates the item if it does not exist
 *    in the database (in other words, if the new item number is different
 *    than single existing number); replaces the old item in memory
 *
 * Update item quantity (Post): updates the item quantity only if the item
 *    number matches the only item in the database
 *
 * Events: only sees if the single item is below reorder quantity, but
 *    can generate the one event per checkForEvents() call
 *
 * Because the item is stored only in memory, the single item will not be
 * persisted if you shut down the Adapter Server, unlike the real JDBC
 * implementation.
 *
 * All properties except for the reorder_quantity (in other words, the JDBC
 * properties) will be ignored.
 *
 * To see modifications from the generated Java implmentation, look for
 * comments of the form:
 *
 *    // XXX added for TestingInventoryAdapter
 */
/**
```

```
 * This is an example Partner Agreement Manager Adapter for a simple Inventory system.
    It
 * demonstrates Get and Post operations as well as event polling.
 *
 * This adapter should be accompanied by the Example_Inventory_Item.1 BO.
 * This adapter also makes use of the Operation_Status.1 BO.  Before running
 * any operations, be sure to set up your simple Inventory system by running
 * the accompanying SQL schema script to create the Inventory database
 * table(s).
 *
 */
public class TestingInventoryAdapter extends Adapter
{
   static final String FileVersion = "$Revision: 9 $";

   //-----------------------------------------------------------------------
   // Constants - Protected - Internal
   //-----------------------------------------------------------------------

   protected static final String UNKNOWN_OPERATION_TYPE_ID =
      "Trying to execute an Adapter operation with an unknown operation type";
   protected static final String UNKNOWN_OPERATION_ID =
      "Trying to execute an unknown Adapter operation";


   //-----------------------------------------------------------------------
   // Constants - Protected - Adapter Properties
   //-----------------------------------------------------------------------

   protected static final String PROP_REORDER_QUANTITY = "reorder_quantity";
   protected static final String PROP_JDBC_DRIVER = "jdbc_driver";
   protected static final String PROP_PASSWORD = "password";
   protected static final String PROP_DATABASE_URL = "database_url";
   protected static final String PROP_USER = "user";


   //-----------------------------------------------------------------------
   // Constants - Protected - Adapter Events
   //-----------------------------------------------------------------------

   protected static final String EVENT_EXAMPLE_INVENTORY_ITEM_BELOW_REORDER_QUANTITY =
     "Example inventory item below reorder quantity";


   //-----------------------------------------------------------------------
   // Constants - Private - Post Operations
   //-----------------------------------------------------------------------

   // "Create inventory item" constants
   protected static final String CREATE_INVENTORY_ITEM = "Create inventory item";

   // "Update item quantity" constants
   protected static final String UPDATE_ITEM_QUANTITY = "Update item quantity";


   //-----------------------------------------------------------------------
   // Constants - Private - Get Operations
   //-----------------------------------------------------------------------

   // "Get inventory information" constants
   protected static final String GET_INVENTORY_INFORMATION = "Get inventory
     information";
   protected static final String GET_INVENTORY_INFORMATION_INPUT_ITEM_NUMBER =
     "item_number";
```

```
//------------------------------------------------------------------------
// Variables - Protected - Adapter Properties
//------------------------------------------------------------------------

/**
 * The on hand quantity below which items should be reordered.
 */
protected int reorder_quantity;

/**
 * The Java class name of the database driver.
 */
protected String jdbc_driver;

/**
 * The user's password.
 */
protected String password;

/**
 * A database URL of the form jdbc:subprotocol:subname .
 */
protected String database_url;

/**
 * The database user on whose behalf the Connection is being made.
 */
protected String user;

// XXX added for TestingInventoryAdapter
/**
 * These four fields represent the simulated inventory database of a
 * single item.  They are named after the fields of the
 * Example_Inventory_Item.1 Business Object.
 */
private int item_number = Integer.MIN_VALUE;
// the item number value before any item has been created
private String description;
private int quantity_on_hand;
private String warehouse_id;


//------------------------------------------------------------------------
// Constructors
//------------------------------------------------------------------------

/**
 * Default constructor.
 */
public TestingInventoryAdapter() {
   super();
}


//------------------------------------------------------------------------
// Methods - Lifecycle
//------------------------------------------------------------------------

/**
 * Called when the adapter is started for the first time.
 *
 * @param context Holds Adapter specific properties, may be empty.
 *
 * @exception ISException Thrown if there is a problem starting the
 * Adapter.
 */
public void startup(AdapterContext context)
```

```
   throws ISException {

   // load the properties from the adapter
   loadAdapterProperties(context);

   // add startup code here
}


/**
 * Called when the adapter is stopped. Allows the adapter to perform any
 * necessary cleanup work.
 *
 * @param context Holds Adapter specific properties, may be empty.
 *
 * @exception ISException Thrown if there is a problem starting the
 * Adapter.
 */
public void shutdown(AdapterContext context)
   throws ISException {

   // add shutdown code here
}


//-------------------------------------------------------------------------
// Methods - Execution
//-------------------------------------------------------------------------

/**
 * Called to execute a specific operation. This method must be overridden
 * by all Adapter subclasses.
 *
 * This method can be called by multiple threads.
 *
 * If the adapter subclass supports transactions, the transaction will be
 * either commited after this method returns or rolled back if an exception
 * is thrown. If the Adapter subclass does not support transactions, it
 * must either commit or rollback all stateful actions performed during
 * execution of this method.
 *
 * @param context Runtime context for the operation. Container for
 * operation inputs and outputs (output values must be added by the Adapter
 * subclass implementation).
 *
 * @exception ISException Thrown if there is an error during execution.
 */
public void execute(OperationContext context)
   throws ISException {
   int operation_type_id;
   String operation_id;

   operation_type_id = context.getOperationTypeID();
   operation_id = context.getOperationID();

   switch (operation_type_id) {
   case OperationContext.GET:
      if (operation_id.equals(GET_INVENTORY_INFORMATION)) {
         executeGetInventoryInformation(context);
      } else {
         throw new ISException(UNKNOWN_OPERATION_ID + ": " + operation_id);
      }
      break;

   case OperationContext.POST:
      if (operation_id.equals(CREATE_INVENTORY_ITEM)) {
         executeCreateInventoryItem(context);
      } else if (operation_id.equals(UPDATE_ITEM_QUANTITY)) {
         executeUpdateItemQuantity(context);
      } else {
         throw new ISException(UNKNOWN_OPERATION_ID + ": " + operation_id);
      }
      break;

   case OperationContext.ADVANCED:
      break;

   default:
      throw new ISException(UNKNOWN_OPERATION_TYPE_ID + ": " + operation_type_id);
   }
}


/**
 * Retrieves information about an item from the Inventory database and
 * stores it into the output Example_Inventory_Item.1 BO.
 *
 * Returns Operation_Status.result of "success" if item is found and
 * information is successfully queried.
 *
 * Returns Operation_Status.result of "failure" if item_number input field
 * is not an integer.  Also returns "failure" if a database error occurs.
 * Also returns "failure" if no item with the given number exists in the
 * database.
 */
protected void executeGetInventoryInformation(OperationContext context)
   throws ISException {
   String item_number;
   Element status_bo;
   Element output_bo;
   int item; // XXX added for TestingInventoryAdapter

   // load the inputs from the OperationContext
   item_number =
  context.getInputVariant(GET_INVENTORY_INFORMATION_INPUT_ITEM_NUMBER);

   // create the output Business Objects
   output_bo = context.createOutputBOElement();
   status_bo = context.createStatusBOElement();

   // add execute code here

   // XXX added for TestingInventoryAdapter - begin

   // validate input field(s)
   try {
       item = Integer.parseInt(item_number);
   } catch (NumberFormatException e) {
       status_bo.setData("result", "failure");
       status_bo.setData("reason", "Error: item_number input field"
                                 + " is not an integer: " + item_number);
       return;
   }

   // call simulated inventory database
   if (item != this.item_number) {
       status_bo.setData("result", "failure");
       status_bo.setData("reason", "Error: Item " + item_number
                                 + " does not exist in the database.");
       return;
   }

   // set output BO field(s)
   output_bo.setData("item_number",
                                  Integer.toString(this.item_number));
    output_bo.setData("description", this.description);
    output_bo.setData("quantity_on_hand",
                                  Integer.toString(this.quantity_on_hand));
    output_bo.setData("warehouse_id", this.warehouse_id);

    // set status BO field(s)
    status_bo.setData("result", "success");

   // XXX added for TestingInventoryAdapter - end
}


/**
 * Updates Inventory database with a new on-hand quantity using the data in
 * the input Example_Inventory_Item.1 Business Object.  Ignores the
 * description and warehouse_id BO fields.
 *
 * Returns Operation_Status.result of "success" if item quantity is
 * successfully updated.
 *
 * Returns Operation_Status.result of "failure" if item_number or
 * quantity_on_hand input BO fields are not integers.  Also returns
 * "failure" if a database error occurs.  Also returns "failure" if no item
 * with the given number exists in the database.
 */
protected void executeUpdateItemQuantity(OperationContext context)
   throws ISException {
   Element input_bo;
   Element status_bo;
    String item_str, quan_str;  // XXX added for TestingInventoryAdapter
    int item, quan;             // XXX added for TestingInventoryAdapter

   // load the inputs from the OperationContext
   input_bo = context.getInputBOElement();

   // create the output Business Objects
   status_bo = context.createStatusBOElement();

   // add execute code here

   // XXX added for TestingInventoryAdapter - begin

    // read input BO field(s)
    item_str = input_bo.getData("item_number");
    quan_str = input_bo.getData("quantity_on_hand");

    // validate input BO field(s)
    try {
        item = Integer.parseInt(item_str);
    } catch (NumberFormatException e) {
        status_bo.setData("result", "failure");
        status_bo.setData("reason", "Error: item_number input BO field"
                               + " is not a integer: " + item_str);
        return;
    }
    try {
        quan = Integer.parseInt(quan_str);
    } catch (NumberFormatException e) {
        status_bo.setData("result", "failure");
        status_bo.setData("reason", "Error: quantity_on_hand input BO"
                               + " field is not an integer: "
                               + quan_str);
        return;
    }

    // call simulated inventory database
   if (item != this.item_number) {
       status_bo.setData("result", "failure");
       status_bo.setData("reason", "Error: Item " + item_str
                                   + " does not exist in the database.");
       return;
   }
   this.quantity_on_hand = quan;

   // set status BO field(s)
   status_bo.setData("result", "success");

   // XXX added for TestingInventoryAdapter - end
}


/**
 * Creates a new item in the Inventory database with the data in the input
 * Example_Inventory_Item.1 Business Object.
 *
 * Returns Operation_Status.result of "success" if item is successfully
 * created.
 *
 * Returns Operation_Status.result of "failure" if item_number or
 * quantity_on_hand input BO fields are not integers.  Also returns
 * "failure" if a database error occurs.  Also returns "failure" if an item
 * with the given item_number already exists in the database.
 */
protected void executeCreateInventoryItem(OperationContext context)
   throws ISException {
   Element input_bo;
   Element status_bo;
   String item_str, quan_str;  // XXX added for TestingInventoryAdapter
   String desc, ware;          // XXX added for TestingInventoryAdapter
   int item, quan;             // XXX added for TestingInventoryAdapter

   // load the inputs from the OperationContext
   input_bo = context.getInputBOElement();

   // create the output Business Objects
   status_bo = context.createStatusBOElement();

   // add execute code here

   // XXX added for TestingInventoryAdapter - begin

   // read input BO field(s)
   item_str = input_bo.getData("item_number");
   desc     = input_bo.getData("description");
   quan_str = input_bo.getData("quantity_on_hand");
   ware     = input_bo.getData("warehouse_id");

   // validate input BO field(s)
   try {
       item = Integer.parseInt(item_str);
   } catch (NumberFormatException e) {
       status_bo.setData("result", "failure");
       status_bo.setData("reason", "Error: item_number input BO field"
                                   + " is not a integer: " + item_str);
       return;
   }
   try {
       quan = Integer.parseInt(quan_str);
   } catch (NumberFormatException e) {
       status_bo.setData("result", "failure");
       status_bo.setData("reason", "Error: quantity_on_hand input BO"
                                   + " field is not an integer: "
                                   + quan_str);
```

```
        return;
    }

    // call simulated inventory database
    if (item == this.item_number) {
        status_bo.setData("result", "failure");
        status_bo.setData("reason", "Error: Item " + item_str
                             + " already exists in the database.");
        return;
    }
    this.item_number = item;
    this.description = desc;
    this.quantity_on_hand = quan;
    this.warehouse_id = ware;

    // set status BO field(s)
    status_bo.setData("result", "success");

   // XXX added for TestingInventoryAdapter - end
}


//--------------------------------------------------------------------------
// Methods - Runtime - Event Production
//--------------------------------------------------------------------------

/**
 * Called to check for new IS events. Produced events must be added to the
 * EventContext.
 *
 * @param context Context for the check call, functions as container for
 * produced events.
 *
 * @exception ISException Thrown if there is an error checking for the
 * event.
 */
public final void checkForEvents(EventContext context)
   throws ISException {

   // add event production code here

   // XXX added for TestingInventoryAdapter - begin

    // trap for case when no item has been created yet
    if (this.item_number != Integer.MIN_VALUE
         && this.quantity_on_hand < this.reorder_quantity) {
        context.addVariantEvent(
                     EVENT_EXAMPLE_INVENTORY_ITEM_BELOW_REORDER_QUANTITY,
                     Integer.toString(this.item_number),
                     null);
    }

   // XXX added for TestingInventoryAdapter - end
}


//--------------------------------------------------------------------------
// Methods - Private
//--------------------------------------------------------------------------

/**
 * Loads values for declared properties from the AdapterContext.
 */
private void loadAdapterProperties(AdapterContext context)
   throws ISException {

   if (context == null) {
```

```
            return;
         }

         jdbc_driver = context.getPropertyAsString(PROP_JDBC_DRIVER);
         password = context.getPropertyAsString(PROP_PASSWORD);
         database_url = context.getPropertyAsString(PROP_DATABASE_URL);
         user = context.getPropertyAsString(PROP_USER);

         if (context.isPropertyBound(PROP_REORDER_QUANTITY)) {
            reorder_quantity = context.getPropertyAsInt(PROP_REORDER_QUANTITY);
         }
      }
}
```

Inventory adapter implementation

This is the real Inventory Adapter Java Implementation adapter, which connects through JDBC to an SQL database. It requires the InventoryAdapterHelper class which, in turn, requires the JDBC Library from the com.extricity.adapters.example.jdbc Java package.

All of the substantive Inventory database logic is performed by the InventoryAdapterHelper class. Therefore, most of the modifications in this file are single-line calls to methods in the helper class.

Note: The modifications from the generated Java implementation appear in **bold**. You can also look for comments in this form:

```
// XXX added for InventoryAdapter

//---------------------------------------------------------------------------
// Source code auto-generated by:
//
// Adapter Designer
//
// Generated on: Sat Mar 04 17:19:46 PST 2000
//---------------------------------------------------------------------------

package com.extricity.adapters.example.inv;


import com.extricity.adapter.api.*;

import com.extricity.document.api.*;


/*
 * The [Real] Inventory Adapter Java Implementation
 *
 * This is the real adapter which connects through JDBC to a database.
 * It requires the InventoryAdapterHelper class which, in turn, requires the
 * JDBC Library from the com.extricity.adapters.example.jdbc Java package.
 *
 * All of the substantive Inventory database logic is performed by the
 * InventoryAdapterHelper class.  Therefore, most of the modifications in
 * this file are single-line calls to methods in the helper class.
 *
 * To see modifications from the generated Java implmentation, look for
 * comments of the form:
 *
 *    // XXX added for InventoryAdapter
 */
/**
 * This is an example Partner Agreement Manager adapter for a simple inventory system.
    It demonstrates Get and Post operations as well as event polling.
 *
 * This adapter must be accompanied by the Example_Inventory_Item.1 BO.
 * This adapter also makes use of the Operation_Status.1 BO.  Before running
 * any operations, be sure to set up your simple Inventory system by running
 * the accompanying SQL schema script to create the Inventory database
 * table(s).
 *
 */
public class InventoryAdapter extends Adapter
{
   static final String FileVersion = "$Revision: 6 $";

   //------------------------------------------------------------------------
   // Constants - Protected - Internal
   //------------------------------------------------------------------------

   protected static final String UNKNOWN_OPERATION_TYPE_ID =
      "Trying to execute an Adapter operation with an unknown operation type";
   protected static final String UNKNOWN_OPERATION_ID =
      "Trying to execute an unknown Adapter operation";


   //------------------------------------------------------------------------
   // Constants - Protected - Adapter Properties
   //------------------------------------------------------------------------

   protected static final String PROP_REORDER_QUANTITY = "reorder_quantity";
   protected static final String PROP_JDBC_DRIVER = "jdbc_driver";
   protected static final String PROP_PASSWORD = "password";
   protected static final String PROP_DATABASE_URL = "database_url";
   protected static final String PROP_USER = "user";


   //------------------------------------------------------------------------
   // Constants - Protected - Adapter Events
   //------------------------------------------------------------------------

   protected static final String EVENT_EXAMPLE_INVENTORY_ITEM_BELOW_REORDER_QUANTITY =
     "Example inventory item below reorder quantity";


   //------------------------------------------------------------------------
   // Constants - Private - Post Operations
   //------------------------------------------------------------------------

   // "Create inventory item" constants
   protected static final String CREATE_INVENTORY_ITEM = "Create inventory item";

   // "Update item quantity" constants
   protected static final String UPDATE_ITEM_QUANTITY = "Update item quantity";


   //------------------------------------------------------------------------
   // Constants - Private - Get Operations
   //------------------------------------------------------------------------

   // "Get inventory information" constants
   protected static final String GET_INVENTORY_INFORMATION = "Get inventory
     information";
```

```
protected static final String GET_INVENTORY_INFORMATION_INPUT_ITEM_NUMBER =
  "item_number";


//-------------------------------------------------------------------------
// Variables - Protected - Adapter Properties
//-------------------------------------------------------------------------

/**
 * The on hand quantity below which items should be reordered.
 */
protected int reorder_quantity;

/**
 * The Java class name of the database driver.
 */
protected String jdbc_driver;

/**
 * The user's password.
 */
protected String password;

/**
 * A database URL of the form jdbc:subprotocol:subname .
 */
protected String database_url;

/**
 * The database user on whose behalf the Connection is being made.
 */
protected String user;

// XXX added for InventoryAdapter
// The helper class for the Example Inventory Adapter
private InventoryAdapterHelper helper;


//-------------------------------------------------------------------------
// Constructors
//-------------------------------------------------------------------------

/**
 * Default constructor.
 */
public InventoryAdapter() {
   super();

   // XXX added for InventoryAdapter
   this.helper = new InventoryAdapterHelper();
}


//-------------------------------------------------------------------------
// Methods - Lifecycle
//-------------------------------------------------------------------------

/**
 * Called when the adapter is started for the first time.
 *
 * @param context Holds Adapter specific properties, may be empty.
 *
 * @exception ISException Thrown if there is a problem starting the
 * Adapter.
 */
public void startup(AdapterContext context)
   throws ISException {
      // load the properties from the adapter
      loadAdapterProperties(context);

      // add startup code here

      // XXX added for InventoryAdapter
      this.helper.startup(this.jdbc_driver, this.database_url,
                          this.user, this.password, this.reorder_quantity);
   }


   /**
    * Called when the adapter is stopped. Allows the adapter to perform any
    * necessary cleanup work.
    *
    * @param context Holds Adapter specific properties, may be empty.
    *
    * @exception ISException Thrown if there is a problem starting the
    * Adapter.
    */
   public void shutdown(AdapterContext context)
      throws ISException {

      // add shutdown code here

      // XXX added for InventoryAdapter
      this.helper.shutdown();
   }


   //------------------------------------------------------------------------
   // Methods - Execution
   //------------------------------------------------------------------------

   /**
    * Called to execute a specific operation. This method must be overridden
    * by all Adapter subclasses.
    *
    * This method can be called by multiple threads.
    *
    * If the adapter subclass supports transactions, the transaction will be
    * either commited after this method returns or rolled back if an exception
    * is thrown. If the Adapter subclass does not support transactions, it
    * must either commit or rollback all stateful actions performed during
    * execution of this method.
    *
    * @param context Runtime context for the operation. Container for
    * operation inputs and outputs (output values must be added by the Adapter
    * subclass implementation).
    *
    * @exception ISException Thrown if there is an error during execution.
    */
   public void execute(OperationContext context)
      throws ISException {
      int operation_type_id;
      String operation_id;

      operation_type_id = context.getOperationTypeID();
      operation_id = context.getOperationID();

      switch (operation_type_id) {
      case OperationContext.GET:
         if (operation_id.equals(GET_INVENTORY_INFORMATION)) {
            executeGetInventoryInformation(context);
         } else {
            throw new ISException(UNKNOWN_OPERATION_ID + ": " + operation_id);
      }
      break;

   case OperationContext.POST:
      if (operation_id.equals(CREATE_INVENTORY_ITEM)) {
         executeCreateInventoryItem(context);
      } else if (operation_id.equals(UPDATE_ITEM_QUANTITY)) {
         executeUpdateItemQuantity(context);
      } else {
         throw new ISException(UNKNOWN_OPERATION_ID + ": " + operation_id);
      }
      break;

   case OperationContext.ADVANCED:
      break;

   default:
      throw new ISException(UNKNOWN_OPERATION_TYPE_ID + ": " + operation_type_id);
   }
}


/**
 * Retrieves information about an item from the Inventory database and
 * stores it into the output Example_Inventory_Item.1 BO.
 *
 * Returns Operation_Status.result of "success" if item is found and
 * information is successfully queried.
 *
 * Returns Operation_Status.result of "failure" if item_number input field
 * is not an integer.  Also returns "failure" if a database error occurs.
 * Also returns "failure" if no item with the given number exists in the
 * database.
 */
protected void executeGetInventoryInformation(OperationContext context)
   throws ISException {
   String item_number;
   Element status_bo;
   Element output_bo;

   // load the inputs from the OperationContext
   item_number =
  context.getInputVariant(GET_INVENTORY_INFORMATION_INPUT_ITEM_NUMBER);

   // create the output Business Objects
   output_bo = context.createOutputBOElement();
   status_bo = context.createStatusBOElement();

   // add execute code here

   // XXX added for InventoryAdapter
    this.helper.getInventoryInformation(item_number, output_bo,
       status_bo);
}


/**
 * Updates Inventory database with a new on-hand quantity using the data in
 * the input Example_Inventory_Item.1 Business Object.  Ignores the
 * description and warehouse_id BO fields.
 *
 * Returns Operation_Status.result of "success" if item quantity is
 * successfully updated.
 *
 * Returns Operation_Status.result of "failure" if item_number or
 * quantity_on_hand input BO fields are not integers.  Also returns
 * "failure" if a database error occurs.  Also returns "failure" if no item
 * with the given number exists in the database.
 */
protected void executeUpdateItemQuantity(OperationContext context)
   throws ISException {
   Element input_bo;
   Element status_bo;

   // load the inputs from the OperationContext
   input_bo = context.getInputBOElement();

   // create the output Business Objects
   status_bo = context.createStatusBOElement();

   // add execute code here

   // XXX added for InventoryAdapter
   this.helper.updateItemQuantity(input_bo, status_bo);
}


/**
 * Creates a new item in the Inventory database with the data in the input
 * Example_Inventory_Item.1 Business Object.
 *
 * Returns Operation_Status.result of "success" if item is successfully
 * created.
 *
 * Returns Operation_Status.result of "failure" if item_number or
 * quantity_on_hand input BO fields are not integers.  Also returns
 * "failure" if a database error occurs.  Also returns "failure" if an item
 * with the given item_number already exists in the database.
 */
protected void executeCreateInventoryItem(OperationContext context)
   throws ISException {
   Element input_bo;
   Element status_bo;

   // load the inputs from the OperationContext
   input_bo = context.getInputBOElement();

   // create the output Business Objects
   status_bo = context.createStatusBOElement();

   // add execute code here

   // XXX added for InventoryAdapter
   this.helper.createInventoryItem(input_bo, status_bo);
}


//-----------------------------------------------------------------------------
// Methods - Runtime - Event Production
//-----------------------------------------------------------------------------

/**
 * Called to check for new IS events. Produced events must be added to the
 * EventContext.
 *
 * @param context Context for the check call, functions as container for
 * produced events.
 *
 * @exception ISException Thrown if there is an error checking for the
 * event.
 */
public final void checkForEvents(EventContext context)
   throws ISException {
      // add event production code here

      // XXX added for InventoryAdapter
      this.helper.checkForEvents(context);
   }


   //-------------------------------------------------------------------------
   // Methods - Private
   //-------------------------------------------------------------------------

   /**
    * Loads values for declared properties from the AdapterContext.
    */
   private void loadAdapterProperties(AdapterContext context)
      throws ISException {

      if (context == null) {
         return;
      }

      jdbc_driver = context.getPropertyAsString(PROP_JDBC_DRIVER);
      password = context.getPropertyAsString(PROP_PASSWORD);
      database_url = context.getPropertyAsString(PROP_DATABASE_URL);
      user = context.getPropertyAsString(PROP_USER);

      if (context.isPropertyBound(PROP_REORDER_QUANTITY)) {
         reorder_quantity = context.getPropertyAsInt(PROP_REORDER_QUANTITY);
      }
   }
}
```

Inventory adapter helper class

This class contains all the substantive adapter logic so you can regenerate the Adapter Implementation Java source code file freely and often. All you need to do is copy and paste snippets of code from this file into four Java Adapter Implementation sections:

- Variables
- Life cycle Methods
- Execution Methods
- Event Production Methods

Note: The code snippets that you need to copy into the Inventory Adapter code file appear in **bold**. You can also look for comments in this form:

```
/* Copy and paste:
/*
 *
 * The Example Inventory Adapter
 *
 * This is an example Partner Agreement Manager adapter for a simple inventory system.
    It
 * demonstrates Get and Post operations as well as event polling.
 *
 * This adapter should be accompanied by the Example_Inventory_Item.1 BO.
 * This adapter also makes use of the Operation_Status.1 BO.  Be sure to
 * set up your simple Inventory system by running the accompanying
 * SQL schema script to create the Inventory database table(s).
*/

package com.extricity.adapters.example.inv;


import com.extricity.adapters.example.jdbc.JDBCLibrary;

import com.extricity.adapter.api.*;

import com.extricity.is.lib.util.Debug;

import com.extricity.document.api.BusinessObject;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import java.util.Enumeration;
import java.util.Vector;
import java.util.Hashtable;


public class InventoryAdapterHelper
{
   static final String FileVersion = "$Revision: 4 $";

   //------------------------------------------------------------------------
   // Constants
   //------------------------------------------------------------------------

   /** Event names */
   private static final String REORDER_EVENT
          = "Example inventory item below reorder quantity";

   /** Example_Inventory_Item BO fields, database columns */
   private static final String ITEM = "item_number";
   private static final String DESC = "description";
   private static final String QUAN = "quantity_on_hand";
   private static final String WARE = "warehouse_id";

   /** Operation_Status BO fields and field values */
   private static final String RESU = "result";
   private static final String REAS = "reason";
   private static final String SUCC = "success";
   private static final String FAIL = "failure";

   /** Database details */
   private static final String TABLE = "Example_Inventory_Items_tbl";
   private static final String COLUMNS
      = ITEM + ", " + DESC + ", " + QUAN + ", " + WARE;


   //------------------------------------------------------------------------
   // Variables
   //------------------------------------------------------------------------

   /** The database connection. */
   private Connection con;
   private int reorder_quantity;

   /*
    * These "copy and paste" sections should be copied and pasted into the
    * corresponding section in the generated adapter Java source code file.
 * They demonstrate how the helper should be called.
 */

/* Copy and paste:
 *
   // The helper class for the Example Inventory Adapter
   private InventoryAdapterHelper helper;
 *
 */


//-------------------------------------------------------------------------
// Constructors
//-------------------------------------------------------------------------

/* Copy and paste:
 *
   helper = new InventoryAdapterHelper();
 *
 */
/**
 * Default constructor.
 */
public InventoryAdapterHelper() {
}


/* ************************************************************************
 *
 * Section 1: These "entry point" methods should be called from the
 * generated adapter Java source code file.
 *
 * ************************************************************************/

//-------------------------------------------------------------------------
// Methods - Public - Lifecycle Entry Point
//-------------------------------------------------------------------------

/* Copy and paste:
 *
   helper.startup(jdbc_driver, database_url, user, password,
                  reorder_quantity);
 *
 */
/**
 * Opens a database connection and sets reorder quantity.
 *
 * @param jdbc_driver      The Java class name of the database driver.
 * @param database_url     A database url of the form
 *                         jdbc:<em>subprotocol</em>:<em>subname</em> .
 * @param user             The database user on whose behalf the
 *                         Connection is being made.
 * @param password         The user's password.
 * @param reorder_quantity The on hand quantity below which items should
 *                         be reordered.
 *
 * @exception ISException  if a database error occurs.
 */
public void startup(String jdbc_driver, String database_url,
      String user, String password, int reorder_quantity)
   throws ISException {
   this.con = JDBCLibrary.openConnection(jdbc_driver, database_url,
      user, password);
   this.reorder_quantity = reorder_quantity;
}
```

```
/* Copy and paste:
 *
   helper.shutdown();
 *
 */
/**
 * Closes database connection.
 *
 * @exception ISException  if a database error occurs.
 */
public void shutdown()
   throws ISException {
   JDBCLibrary.closeConnection(this.con);

}


//------------------------------------------------------------------------
// Methods - Public - Execution Entry Point
//------------------------------------------------------------------------

/* Copy and paste:
 *
   helper.getInventoryInformation(item_number, output_bo, status_bo);
 *
 */
/**
 * Retrieves information about an item from the Inventory database and
 * stores it into the output Example_Inventory_Item.1 BO.
 *
 * Returns Operation_Status.result of "success" if item is found and
 * information is successfully queried.
 *
 * Returns Operation_Status.result of "failure" if item_number input
 * field is not an integer.  Also returns "failure" if a database error
 * occurs.  Also returns "failure" if no item with the given number
 * exists in the database.
 *
 * @param item_str  The item number as a String.
 *                  This String input must be parsable as integer.
 * @param output_bo BO Type: Extricity.3.Example_Inventory_Item.1
 * @param status_bo BO Type: Extricity.3.Operation_Status.1
 */
public void getInventoryInformation(String item_str,
   BusinessObject output_bo,
   BusinessObject status_bo) {
   int item;
   String info_str;
   Hashtable item_info;

   // validate input field(s)
   try {
      item = Integer.parseInt(item_str);
   } catch (NumberFormatException e) {
      status_bo.setField(RESU, FAIL);
      status_bo.setField(REAS, "Error: " + ITEM
         + " input field is not an integer: "
         + item_str);
      return;
   }

   // call Inventory database API
   try {
      item_info = getItemInfo(item);
   } catch (SQLException e) {
      status_bo.setField(RESU, FAIL);
      status_bo.setField(REAS, "Database error: " + e.toString() + "\n");
```

```
         e.printStackTrace(System.err);
         return;
      }
      if (item_info == null) {
         status_bo.setField(RESU, FAIL);
         status_bo.setField(REAS, "Error: Item " + item_str
            + " does not exist in the database.");
         return;
      }

      // set output BO field(s)
      output_bo.setField(ITEM, item_str);
      output_bo.setField(DESC, (String) item_info.get(DESC));
      output_bo.setField(QUAN, item_info.get(QUAN).toString());
      output_bo.setField(WARE, (String) item_info.get(WARE));

      // set status BO field(s)
      status_bo.setField(RESU, SUCC);
   }


   /* Copy and paste:
    *
      helper.createInventoryItem(input_bo, status_bo);
    */
   /**
    * Creates a new item in the Inventory database with the data in the
    * input Example_Inventory_Item.1 Business Object.
    *
    * Returns Operation_Status.result of "success" if item is successfully
    * created.
    *
    * Returns Operation_Status.result of "failure" if item_number or
    * quantity_on_hand input BO fields are not integers.  Also returns
    * "failure" if a database error occurs.  Also returns "failure" if an
    * item with the given item_number already exists in the database.
    *
    * @param input_bo  BO Type: Extricity.3.Example_Inventory_Item.1
    * @param status_bo BO Type: Extricity.3.Operation_Status.1
    */
   public void createInventoryItem(BusinessObject input_bo,
         BusinessObject status_bo) {
      String item_str, quan_str;
      String desc, ware;
      int item, quan;
      boolean success;

      // read input BO field(s)
      item_str = input_bo.getField(ITEM);
      desc     = input_bo.getField(DESC);
      quan_str = input_bo.getField(QUAN);
      ware     = input_bo.getField(WARE);

      // validate input BO field(s)
      try {
         item = Integer.parseInt(item_str);
      } catch (NumberFormatException e) {
         status_bo.setField(RESU, FAIL);
         status_bo.setField(REAS, "Error: " + ITEM
            + " input BO field is not an integer: "
            + item_str);
         return;
      }
      try {
         quan = Integer.parseInt(quan_str);
      } catch (NumberFormatException e) {
         status_bo.setField(RESU, FAIL);
      status_bo.setField(REAS, "Error: " + QUAN
         + " input BO field is not an integer: "
         + quan_str);
      return;
   }

   // call Inventory database API
   try {
      success = createItem(item, desc, quan, ware);
   } catch (SQLException e) {
      status_bo.setField(RESU, FAIL);
      status_bo.setField(REAS, "Database error: " + e.toString() + "\n");
      e.printStackTrace(System.err);
      return;
   }

   // set status BO field(s)
   if (success) {
      status_bo.setField(RESU, SUCC);
   } else {
      status_bo.setField(RESU, FAIL);
      status_bo.setField(REAS, "Error: Item " + item_str
         + " already exists in the database.");
   }
}


/* Copy and paste:
 *
   helper.updateItemQuantity(input_bo, status_bo);
 */
/**
 * Updates Inventory database with a new on hand quantity using the data
 * in the input Example_Inventory_Item.1 Business Object.  Ignores the
 * description and warehouse_id BO fields.
 *
 * Returns Operation_Status.result of "success" if item quantity is
 * successfully updated.
 *
 * Returns Operation_Status.result of "failure" if item_number or
 * quantity_on_hand input BO fields are not integers.  Also returns
 * "failure" if a database error occurs.  Also returns "failure" if no
 * item with the given number exists in the database.
 *
 * @param input_bo  BO Type: Extricity.3.Example_Inventory_Item.1
 * @param status_bo BO Type: Extricity.3.Operation_Status.1
 */
public void updateItemQuantity(BusinessObject input_bo,
      BusinessObject status_bo) {
   String item_str, quan_str;
   int item, quan;
   boolean success;

   // read input BO field(s)
   item_str = input_bo.getField(ITEM);
   quan_str = input_bo.getField(QUAN);

   // validate input BO field(s)
   try {
      item = Integer.parseInt(item_str);
   } catch (NumberFormatException e) {
      status_bo.setField(RESU, FAIL);
      status_bo.setField(REAS, "Error: " + ITEM
         + " input BO field is not a integer: "
         + item_str);
      return;
   }
```

```
         try {
            quan = Integer.parseInt(quan_str);
         } catch (NumberFormatException e) {
            status_bo.setField(RESU, FAIL);
            status_bo.setField(REAS, "Error: " + QUAN
               + " input BO field is not an integer: "
               + quan_str);
            return;
         }

         // call Inventory database API
         try {
            success = updateItemQuantity(item, quan);
         } catch (SQLException e) {
            status_bo.setField(RESU, FAIL);
            status_bo.setField(REAS, "Database error: " + e.toString() + "\n");
            e.printStackTrace(System.err);
            return;
         }

         // set status BO field(s)
         if (success) {
            status_bo.setField(RESU, SUCC);
         } else {
            status_bo.setField(RESU, FAIL);
            status_bo.setField(REAS, "Error: Item " + item_str
               + " does not exist in the database.");
         }
      }


      //-------------------------------------------------------------------------
      // Methods - Public - Event Production Entry Point
      //-------------------------------------------------------------------------

      /* Copy and paste:
       *
          helper.checkForEvents(context);
       *
       */
      /**
       * Called to check for new IS events. Produced events must be added to the
       * EventContext.
       *
       * @param context Context for the check call, functions as container for
       * produced events.
       *
       * @exception ISException  if there is an error checking for the
       * event.
       */
      public void checkForEvents(EventContext context)
         throws ISException {
         Enumeration enum;

         try {
            enum = getReorderItems(this.reorder_quantity);
         } catch (SQLException e) {
            throw new ISException("Database error when checking for events", e);
         }
         while (enum.hasMoreElements()) {
            context.addISEvent(REORDER_EVENT, enum.nextElement().toString(),
               null);
   }
      }


      /* ***********************************************************************
```

```
 *
 * Section 2: The following Inventory database methods are not directly
 * related to Partner Agreement Manager adapters.  They only represent API calls
 into the
 * simple inventory database.
 *
 * Disclaimer: In the interests of simplicity, a few corner cases are
 *             not handled.  These could easily be trapped in the
 *             Execution Entry Point methods above.
 *
 *             - methods allows negative item numbers and quantities
 *
 *             - methods do not report truncation of input fields if they
 *               are wider than database columns
 *
 * *********************************************************************/

//---------------------------------------------------------------------------
// Methods - Private - Inventory database query (Accessors)
//---------------------------------------------------------------------------

/**
 * Determines if an item already exists in the Inventory database.
 *
 * @param item The number of the item that may or may not exist.
 *
 * @return True if item number already exists.  False if it does not.
 *
 * @exception SQLException  if a database error occurs.
 */
private boolean itemExists(int item)
   throws SQLException {
   String sql;
   PreparedStatement ps = null;
   ResultSet rs = null;
   int rows = 0;

   sql = "SELECT COUNT(*) FROM " + TABLE + " WHERE " + ITEM + " = ?";
   if (Debug.ON) {
      System.out.println("itemExists: " + sql + "\n");
   }

   try {
      ps = this.con.prepareStatement(sql);
      ps.setInt(1, item);
      rs = ps.executeQuery();
      if (rs.next()) {
      rows = rs.getInt(1);  // get the count column (first position)
      }
   } finally {
      JDBCLibrary.forceQueryClose(rs, ps);
   }

   return (rows > 0);
}


/**
 * Retrieves information about an item from the Inventory database.
 *
 * @param item The queried item number.
 *
 * @return Hashtable with column names as keys and column data as Integer
 *         or String object values.  (See schema file for names of columns
 *         in Example_Inventory_Items_tbl.)  Null if item does not exist.
 *
 * @exception SQLException  if a database error occurs.
 */
private Hashtable getItemInfo(int item)
      throws SQLException {
   String sql;
   PreparedStatement ps = null;
   ResultSet rs = null;
   int quan;
   String desc, ware;
   Hashtable item_info = new Hashtable();

   sql = "SELECT " + COLUMNS + " FROM " + TABLE + " WHERE " + ITEM
      + " = ?";
   if (Debug.ON) {
      System.out.println("getItemInfo: " + sql + "\n");
   }

   try {
      ps = this.con.prepareStatement(sql);
      ps.setInt(1, item);
      rs = ps.executeQuery();
      if (rs.next()) {
         Debug.assert(item == rs.getInt(1));
         desc = rs.getString(2);
         quan = rs.getInt(3);
         ware = rs.getString(4);
      } else {
         return (null);
         }
   } finally {
      JDBCLibrary.forceQueryClose(rs, ps);
   }

   // put key-value pairs into Hashtable
   item_info.put(ITEM, new Integer(item));
   item_info.put(DESC, desc);
   item_info.put(QUAN, new Integer(quan));
   item_info.put(WARE, ware);
   return (item_info);
}


/**
 * Returns a list of all items.
 *
 * @return Enumeration of item numbers as Integer objects.
 *
 * @exception SQLException  if a database error occurs.
 */
private Enumeration getAllItems()
      throws SQLException {
   String sql;
   PreparedStatement ps = null;
   ResultSet rs = null;
   Vector items = new Vector();

   sql = "SELECT " + ITEM + " FROM " + TABLE;
   if (Debug.ON) {
      System.out.println("getAllItems: " + sql + "\n");
   }

   try {
      ps = this.con.prepareStatement(sql);
      rs = ps.executeQuery();
      while (rs.next()) {
         items.addElement(new Integer(rs.getInt(1)));
         }
   } finally {
         JDBCLibrary.forceQueryClose(rs, ps);
      }

      return (items.elements());
   }


   /**
    * Returns a list of all items whose on hand quantity is below the
    * reorder quantity.
    *
    * @param quan The on hand quantity below which items should be reordered.
    *
    * @return Enumeration of item numbers as Integer objects.
    *
    * @exception SQLException  if a database error occurs.
    */
   private Enumeration getReorderItems(int quan)
         throws SQLException {
      String sql;
      PreparedStatement ps = null;
      ResultSet rs = null;
      Vector items = new Vector();

      sql = "SELECT " + ITEM + " FROM " + TABLE + " WHERE " + QUAN + " < ?";
      if (Debug.ON) {
         System.out.println("getReorderItems: " + sql + "\n");
      }

      try {
         ps = this.con.prepareStatement(sql);
         ps.setInt(1, quan);
         rs = ps.executeQuery();
         while (rs.next()) {
            items.addElement(new Integer(rs.getInt(1)));
            }
      } finally {
         JDBCLibrary.forceQueryClose(rs, ps);
      }

      return (items.elements());
   }


   //------------------------------------------------------------------------
   // Methods - Private - Inventory database update (Mutators)
   //------------------------------------------------------------------------

   /**
    * Creates a new item in the Inventory database.
    *
    * @param item A new item number.
    * @param desc The item's description.
    * @param quan The item's initial quantity on hand.
    * @param ware The item's warehouse identifier.
    *
    * @return True if item was successfully created.  False if item already
    *         exists.
    *
    * @exception SQLException  if a database error occurs.
    */
   private synchronized boolean createItem(int item, String desc,
      int quan, String ware)
      throws SQLException {
      String sql;
      PreparedStatement ps = null;
      ResultSet rs;
   int rows = 0;

   // make sure item doesn't already exist
   if (itemExists(item)) {
      return (false);
   }

   sql = "INSERT INTO " + TABLE + " (" + COLUMNS + ") values (?, ?, ?, ?)";
   if (Debug.ON) {
      System.out.println("createItem: " + sql + "\n");
   }

   try {
      ps = this.con.prepareStatement(sql);
      ps.setInt(1, item);
      ps.setString(2, desc);
      ps.setInt(3, quan);
      ps.setString(4, ware);
      rows = ps.executeUpdate();
      this.con.commit();
   } finally {
      JDBCLibrary.forceUpdateClose(ps);
   }

   if (Debug.ON) {
         System.out.println("createItem: " + rows + " row(s) inserted.");
   }
   return (true);
}


/**
 * Updates Inventory database with a new on hand quantity.
 *
 * @param item The number of the item to update.
 * @param quan The new quantity on hand.
 *
 * @return True if quantity is updated successfully.  False if item does
 *         not exist.
 *
 * @exception SQLException  if a database error occurs.
 */
private synchronized boolean updateItemQuantity(int item, int quan)
      throws SQLException {
   String sql;
   PreparedStatement ps = null;
   int rows;

   // make sure item already exists
   if (!itemExists(item)) {
      return (false);
   }

   sql = "UPDATE " + TABLE + " SET " + QUAN + " = ? WHERE " + ITEM
      + " = ?";
   if (Debug.ON) {
      System.out.println("updateItemQuantity: " + sql + "\n");
   }

   try {
      ps = this.con.prepareStatement(sql);
      ps.setInt(1, quan);
      ps.setInt(2, item);
      rows = ps.executeUpdate();
      this.con.commit();
   } finally {
      JDBCLibrary.forceUpdateClose(ps);
   }

   if (Debug.ON) {
      System.out.println("updateItemQuantity: " + rows
         + " row(s) updated.");
   }
   return (true);
}


//-------------------------------------------------------------------------
// Methods - Public Static - Testing
//-------------------------------------------------------------------------

/**
 * Useful for testing from the command prompt.
 *
 * USAGE:
 *
 * To get a list of all item numbers:
 *   AdapterHelper <driver> <url> <user> <pass>
 *
 * To see if inventory item exists:
 *   AdapterHelper <driver> <url> <user> <pass> <item>
 *
 * To get inventory item info:
 *   AdapterHelper <driver> <url> <user> <pass> <item>
 *
 * To get a all items below a reorder quantity:
 *   AdapterHelper <driver> <url> <user> <pass> <quan>
 *
 * To update inventory item quantity:
 *   AdapterHelper <driver> <url> <user> <pass> <item> <quan>
 *
 * To create an inventory item:
 *   AdapterHelper <driver> <url> <user> <pass> <item> <desc> <quan> <ware>
 *
 * For more information on the first four arguments, see the JDBCLibrary:
 * com.extricity.adapters.example.jdbc.JDBCLibrary .
 */
public static void main(String args[])
        throws ISException, SQLException {
   InventoryAdapterHelper helper;
   Enumeration enum;

   if (args.length < 4) {
      System.err.println("USAGE:"
         + "\n\nTo get a list of all item numbers:"
         + "\n  AdapterHelper <driver> <url>"
         + " <user> <pass>"

         + "\n\nTo see if inventory item exists:"
         + "\n  AdapterHelper <driver> <url>"
         + " <user> <pass> <item>"

         + "\n\nTo get inventory item info:"
         + "\n  AdapterHelper <driver> <url>"
         + " <user> <pass> <item>"

         + "\n\nTo get all items below a reorder quantity:"
         + "\n  AdapterHelper <driver> <url>"
         + " <user> <pass> <quan>"

         + "\n\nTo update inventory item quantity:"
         + "\n  AdapterHelper <driver> <url>"
         + " <user> <pass> <item> <quan>"
```

```
               + "\n\nTo create an inventory item:"
               + "\n  AdapterHelper <driver> <url>"
               + " <user> <pass> <item> <desc> <quan> <ware>");
            return;
         }

         // Lifecycle
         helper = new InventoryAdapterHelper();
         helper.startup(args[0], args[1], args[2], args[3], 0);

         // Execution
         if (args.length == 4) {
            // <driver> <url> <user> <pass>
            enum = helper.getAllItems();
            System.out.println("getAllItems: ");
            while (enum.hasMoreElements()) {
               System.out.println(enum.nextElement().toString());
            }

         } else if (args.length == 5) {
            // <driver> <url> <user> <pass> <item>
            System.out.println("itemExists: "
               + helper.itemExists(Integer.parseInt(args[4])));

            // <driver> <url> <user> <pass> <item>
            System.out.println("getItemInfo: "
               + helper.getItemInfo(Integer.parseInt(args[4])));

            // <driver> <url> <user> <pass> <quan>
            enum = helper.getReorderItems(Integer.parseInt(args[4]));
            System.out.println("getReorderItems: ");
            while (enum.hasMoreElements()) {
               System.out.println(enum.nextElement().toString());
            }

         } else if (args.length == 6) {
            // <driver> <url> <user> <pass> <item> <quan>
            System.out.println("updateItemQuantity: "
               + helper.updateItemQuantity(
                  Integer.parseInt(args[4]),
                  Integer.parseInt(args[5])));
         } else if (args.length == 8) {
            // <driver> <url> <user> <pass> <item> <desc> <quan> <ware>
            System.out.println("createItem: "
               + helper.createItem(Integer.parseInt(args[4]),
               args[5],
               Integer.parseInt(args[6]),
               args[7]));
         }

         // Lifecycle
         helper.shutdown();
      }

   }
```

APPENDIX

B

NOTICES

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
DB2
IBM
MQSeries
SupportPac
WebSphere

Pentium is a registered trademark of Intel Corporation in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

GLOSSARY

action—a task performed as part of a private process. A private process action is the equivalent of a step in a public process. See the following terms in this glossary for more information about the action types you can include in a private process:

- approval action
- extension action
- mapping action
- notification action
- output object action
- script action
- subprocess action
- termination action
- timer action

See also *private process*.

adapter—the software bridge between Partner Agreement Manager processes and specific end-system and business-application interfaces. Adapters manage interactions between business applications and the Adapter Server. They allow private processes to interact with external business applications while a process is running, and they allow PAM to start public processes based on events that occur in external business applications. See also *adapter implementation*, *adapter instance*, *adapter type*.

adapter implementation—the implementation declaration for an adapter type. It specifies the name and location of the Java source file that defines the application logic used to communicate with a specific end system through that end system's interface. The application logic is specified in the form of properties. See also *adapter*, *adapter instance*, *adapter type*.

adapter instance—an instance of an adapter implementation. The adapter instance is used in a private process extension action and provides the specific values to be used for the properties declared in the adapter implementation. See also *adapter*, *adapter implementation*, *adapter type, extension action*.

adapter type—a definition that is stored in XML format and specifies the adapter's properties as well as the operations and events it supports. A single adapter type can have several implementations, and each implementation can have several instances. See also *adapter*, *adapter implementation*, *adapter instance*.

approval action—a private process action that you use to ask for a response from a user before letting the process continue to run. You can use an approval action, for example, to ask for an OK when a purchase order exceeds a predetermined amount. See also *private process*.

business object—a message transmitted as part of a public process. Business objects take the form of purchase orders, acknowledgments, requests for clarification, and so on. See also *business object type*.

business object type—a definition that determines the types of information a message can contain. It has three properties: the top-level element in its element definition set, its key field, and whether instances of it return audit information for non-repudiation purposes. The name of the business object type is the name of the element you select as its top-level element. See also *business object*, *element definition set*, *non-repudiation*.

business object variable—one of the two types of variables used in Partner Agreement Manager to store information within a process. Business object variables create an instance of a business object type. They can be used to store, for example, the outputs from extension actions, the inputs for map actions, or the inputs and outputs for subprocesses. See also *business object*, *business object type*, *extension action*, *variant variable.*

CA—see *certificate authority*.

certificate—a security document that binds a public encryption key to an entity (an individual or organization) known as the principal. The security document (a digital certificate) is signed by another entity known as the issuer. A digital certificate for which both the principal and issuer are the same entity is known as a self-signed certificate. A certificate for which the principal and issuer are different entities is issued by a certificate authority (CA) like VeriSign and is known as a CA-issued (or third-party-signed) certificate. Partner Agreement Manager supports both self-signed and CA-issued certificates. PAM also supports the binding of certificates to be used for signature authentication, message encryption, and SSL authentication for channels other than PAM. See also *certificate authority*, *SSL*.

certificate authority—a trusted third-party organization or company that issues digital certificates used to create digital signatures and public-private key pairs. The role of the certificate authority, or CA, is to authenticate the entities (individuals or organizations) involved in electronic transactions. CAs are a critical component in data security and electronic commerce because they guarantee that the two parties exchanging information are really who they claim to be. See also *certificate*.

channel—a communications mechanism that encapsulates all the processing information needed to send messages to a partner's system, as well as to translate data received from a partner into Partner Agreement Manager messages. PAM provides channels for RosettaNet, EDI, cXML, and other systems and protocols. See also *message*.

digital certificate—see *certificate*.

DTD—Document Type Definition. A type of file associated with SGML and XML documents that defines how the formatting tags should be interpreted by the application presenting the document. In Partner Agreement Manager, a DTD file contains the complete description of a business object type's element definition set. See also *business object*, *business object type*, *element definition set*.

element definition set—a collection of data fields (or elements) or groups of data fields that defines the structure and meaning of a business object type. See also *business object*, *business object type*.

encryption certificate—see *certificate*.

event—a piece of information that comes into Partner Agreement Manager as a message from another source (an enterprise system or business application, for example) and triggers a public process. See also *message*.

event push—a method that uses the HTTP POST mechanism to push events into Partner Agreement Manager as a way to trigger processes. A port on the Process Server is set to listen for events in the form of HTTP POST messages. When a message is detected, PAM uses the information in the message to generate an event. See also *event*.

extended enterprise—a business model under which companies that work together as partners function as efficiently as a single organization through the implementation of automated communication technologies.

extension action—a private process action that communicates via an adapter with an external application that is registered with Partner Agreement Manager. You can use an extension action, for example, to launch a spreadsheet application, perform calculations, and update the enterprise system, or to get information from an enterprise system or listen for an event in the enterprise system. See also *adapter*, *private process*.

LDAP—Lightweight Directory Access Protocol. LDAP provides a standard method for accessing information from a central directory. After user authentication is set up in the LDAP directory, applications that use the LDAP protocol can retrieve the information from that directory. An authenticated user can log in to any application that supports the LDAP protocol with the same user name and password.

linked certificate—see *certificate*.

map—a Java Script or VBScript that inserts data into fields in an output business object type generated by a private process. The map specifies which fields in the output business object type receive data, and it identifies the information source.

map method—a reusable logical block of code that inserts data into a particular type of element or element sequence in a business object type. Within a map method, you can write the expressions that map individual input and output fields in the sequence. Or you can create a submap and drag input fields to output fields and have Partner Agreement Manager create the appropriate mapping expressions. See also *map*, *submap*.

mapping action—a private process action that you use to call a map. The map specifies the fields in a business object type that will receive data extracted from another source. You use a mapping action when you want to extract data from one business object type and insert it in a different business object type. For example, you use a mapping action to transform a purchase order generated by your inventory system into a sales order in a format that your partner expects. See also *map*, *private process*.

message—a structured communication used to pass information and control to another partner in a public process. The action in the process passes to the partner who receives the message. The content of a message is determined by its business object type. A message can be transmitted via synchronous or asynchronous methods, as determined by its communication service type. See *business object type*.

non-repudiation—a business object security feature that authenticates instances of a business object type and maintains an audit record to verify that they were received by the intended recipient. For business object instances that you receive, Partner Agreement Manager authenticates each instance and maintains an audit record to verify that the instance actually originated with the stated originator. If you disable auditing for a business object type, non-repudiation support is disabled for all messages that contain instances of that business object type.

notification action—a private process action that you use to send an e-mail, fax, or pager message to addressees that you specify. You use a notification action to inform someone inside or outside your organization that an event has occurred. For example, you can use a notification action to alert the order entry department when a purchase order arrives from a customer. See also *private process*.

output object action—a private process action that you use to bind a business object to the expected output object and path in a public process. You use an output object action at the point in a private process when you are ready to send a business object to the associated public process. This is typically the last action in the private process. See also *private process*.

partner group—a group of partners that perform the same role in a process at different times. Instead of duplicating a public process and substituting a different partner name, you can set up a partner group for the public process and then designate a specific partner as the participant when you start an instance of the process. For example, you might design a generic purchasing process that works equally well with any of your suppliers and then designate the appropriate partner when you start the process.

partner profile—information that identifies an organization, specifies a contact person in that organization, lists the communication services the organization supports, and defines the organization's security profile. When partners agree to participate in a public process, they must exchange profile information as a way to ensure authenticity before they can proceed.

PIP—Partner Interface Process. RosettaNet PIPs are specialized system-to-system XML-based dialogs that define business processes between supply-chain partners and provide models and documents for the implementation of e-commerce standards. Each PIP includes a technical specification based on the RosettaNet Implementation Framework (RNIF), a message guideline document with a PIP-specific version of the business dictionary, and an XML message guideline document. See also *RosettaNet*.

post method—the last block of code that is executed when a mapping action runs. Its only parameter is the output business object. You use the post method when you need to perform post-processing on the output business object. For example, you might use the post method to set the value of a summary field based on the number of line items in the output business object, or to examine a range of dates in a repeated group, extract the most recent date, and post that date in a header field. See also *mapping action*, *pre method*.

pre method—the first block of code that is executed when a mapping action runs. The pre method's parameters are the map inputs. You use the pre method to access a map's inputs and set global variables based on their content. See also *mapping action*, *post method*.

private process—a task or set of tasks that business partners participating in a public process perform at points where they need to take action internally. Partners participating in a public process must implement a private process for each public process step that they own. A private process begins with input from the public process and ends with output that feeds back into the public process. The input can be the receipt of a business object from a partner, or it can be a triggering event from an internal system. The output is the business object that transfers control back to the public process. See also *action*, *process*, *public process*.

private process action—see *action*.

process—the flow of actions and the exchange of business information between partners in an extended enterprise. A process operates on two levels, public and private. See *extended enterprise*, *private process*, *public process*.

public process—the step-by-step flow of messages, events, and actions between two or more business partners. Public processes are set up by agreement between partners, and each step in a public process has a private process associated with it. A public process is developed by one partner, and all the partners who participate in it must review and approve it before it can be implemented. The partner who designs a public process is its owner. See also *private process*, *process*.

RosettaNet—a consortium of major information technology, electronic components, and semiconductor manufacturing companies that is working to create and implement industry-wide, open e-business process standards. See also *PIP*.

script action—a private process action that consists of a script written in VBScript or JavaScript and is designed to manipulate information or set up conditional actions based on input. You use a script to establish decision-making criteria for branches or loops, to set variables, or to calculate values that are used elsewhere in the private process. See also *private process*.

security certificate—see *certificate*.

self-signed certificate—see *certificate*.

signature certificate—see *certificate*.

SSL—Secure Sockets Layer. The SSL protocol is a security protocol that provides for communications privacy and reliability over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

submap—a secondary level map that is called by a map method to insert data into an output element other than the top-level element. See *map*, *map method*.

subprocess action—a private process action you use to call an existing public process. You can call any public process in which your organization owns the first partner action. For example, you can use a subprocess to get a quote approved by a third-party supplier before responding to a customer. See also *private process*.

termination action—a private process action that you use to stop a process at a predetermined point for a reason that you specify. You can use a termination action to deal with errors in data that might prevent a process from completing successfully. For example, you might want to stop a process in cases where an enterprise system passes incomplete or corrupted information to it. See also *private process*.

third-party-signed certificate—another name for a CA-issued certificate. See *certificate*.

timer action—a private process action that you use to insert a pause. You can use a timer action to specify the period of time you want to elapse before the next action in the process starts. See also *private process*.

variant variable—single field variables. Variant variables store text strings—the type of information contained in a single field element. You can use variant variables to store the input for actions, to set flags (such as the time-out flag for an approval action), to move information within scripts, or to store the results of an approval action. See also *business object variable*.

Index