WebSphere® Partner Agreement Manager

API Guide

Version 2 Release 1

Note: Before using this information and the product it supports, read the information in *Notices* on page 49.

TABLE OF CONTENTS

WELCOME TO THE API GUIDE

This document describes the principles behind WebSphere® Partner Agreement Manager External API, and explains how to use the External API to access the Process Manager functionality in your application. See also the Javadoc for the External API.

To access the Process Manager from your application:

- Read *Using the External API* on page 1 for an overview of how the External API works.

- See *Connecting to the B2B Engine* on page 9 for an example of how to use the External API in your application to connect and log on to the B2B Engine.

- See *Auditing processes* on page 15 for an example of how to get information about processes in the B2B Engine.

- See *Starting processes* on page 23 for an example of how to find process instances, create new process instances, set input, and start processes with the External API.

- See *Using events with processes* on page 31 for an example of how to use the External API to start processes using events.

- See *Using the Document API* on page 39 for an example of how to create a new ElementDefinitionSet and BusinessObjectType from the ExampleXML.dtd file.
- See *Using the External API with adapters* on page 43 for an example of how to use the External API to manage adapters.

Who should use this information

This information is for Java developers who need to use the functionality of the Process Manager in their applications.

Related information

For additional information see the following:

- The readme.txt 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.txt file is located in the root directory of the Partner Agreement Manager installation.

- The index.html file. This file contains links to the Partner Agreement Manager External API readme.txt file and the *Partner Agreement Manager Installation Guide*. Before installation, the index.html file is located in the root directory of the product CD-ROM. After installation, the index.html file is located in the root directory of the Partner Agreement Manager installation.

- The *Partner Agreement Manager Installation Guide*, form number GC34-5964-00, which describes how to install Partner Agreement Manager.

- The *Partner Agreement Manager Administrator's Guide*, form number BIAAAB00, 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 BIAAAC00, 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 BIAAAD00, 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 BIAAAE00, which describes how to write scripts used in Partner Agreement Manager private processes and elsewhere.

- The *Partner Agreement Adapters for MQSeries User's Guide*, form number BIAAAG00, which describes how to install, configure, and run the Partner Agreement Manager Adapters for MQSeries.

- The *Partner Agreement View User's Guide*, form number GC34-5965-00, 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 2.1 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 2.1 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 2.1. 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 2.1 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 2.1 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 2.1 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 will use 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.

1

Using the External API

The External API allows you to write applications that access the functionality of the B2B Engine.

This chapter includes these sections:

About the External API

The External API consists of several packages:

This package	Contains interfaces and classes that do this
com.extricity.api	Manage contacts and contact lists, abstract and root interfaces for sessions, queries. You will not use these abstract interfaces directly.
com.extricity.adapter.api	Manage adapters types, implementations and instances.
com.extricity.admin.api	Manage users and certificates.
com.extricity.api.rmi	Provide connectivity to the B2B engine using remote method invocation (RMI).
com.extricity.document.api	Manage business objects and element definition sets.
com.extricity.partner.api	Manage partners and channels.
com.extricity.api.process	Manage processes.

How the External API works

The External API allows applications to programmatically control the B2B Engine.

The client application creates a session connected to a B2B Engine and then logs in as a specific user. The permissions of the user dictate what operations and access level are allowed. Once the client application has logged in, it may obtain one or more services, which provide the functionality to control some aspect of the B2B Engine.

For a complete description of all the External API components, see the Javadoc for the External API.

Using sessions

A session is a connection to the B2B Engine. Every application that uses the External API will connect to the B2B Engine. A session is a connection, complete with authentication. You must have a session before you use the External API to access the B2B Engine in your application. The example *Connecting to the B2B Engine* on page 9 shows how to create a connection to the B2B Engine.

Using services

Once you have a connection, getting services is typically the next step in using the External API in your application. Services are how you use the functionality of the B2B Engine. There are services associated with each general category of functionality provided by the External API. For example, the ProcessService offers process management services such as creating a new instance of a public process of a given PublicProcessType, or creating, modifying and deleting EventTypes. Likewise PartnerService offers partner management services and DocumentService offers business object management services. See the Javadoc for a complete listing and description of each of the available services and their functions.

To use a service:
- Get the service from the service factory.
- Call methods on the service.

Getting the service from the service factory

Factories provide access to services. In order to use a service, you must first get the service from the service factory. The following pseudo code fragment shows how to use a factory to get a service. Replace occurrences of *xxx* with the appropriate package name and *XXX* with the appropriate service name for one of the supported services. Packages are listed in *About the External API* on page 2. DocumentService, ProcessService, and AdapterService are examples of supported services.

```
import com.extricity.api.*;
import com.extricity.api.rmi.*;
import com.extricity.xxx.api.*;

Session session;
XXXServiceFactory service_factory;
XXXService service;

// Get the service factory for the required service
service_factory = XXXServiceFactory.getServiceFactory();

// Get the required service.
service = service_factory.getService(session);

//Call methods on the service.
service.someMethod();
```

Using queries

When you wish to search for one or more items matching particular criteria, use a query object. Fill in the elements of this object to specify your criteria. Then use the service to perform the query, using the query object to get a Collection of matching objects.

For example, suppose you want a list of partners that match a particular name, for example, "Acme Engineering." To get this list, create a PartnerQuery object and fill in the name element. Once you have the PartnerQuery filled in, you use the PartnerService method getPartners to retrieve a Collection of all partners that match your criteria.

```
//Create the Session & log in (code omitted)
//Get the PartnerService (code omitted)
```

```
// variable declarations
String partnername;
Collection partners;

// Create a new PartnerQuery object
PartnerQuery ();

//Set the name
partnername = "Acme Engineering";
partnerQuery.setName (partnername);

// Use the query object and the partner service to get a
// collection of partners that match the criteria
try {
  partners = myPartnerService.getPartners (PartnerQuery);
}
catch {
  // catch the exceptions and handle the errors
}
```

The External API offers many query objects. See the Javadoc included in your distribution for a description of each query object.

Using the examples

The examples included with this External API are self-contained. However, later examples make reference to earlier ones. So if you are unfamiliar with the B2B Engine, start at the first example and work your way through to the last.

Before you run the examples

Before you run this example, you must configure your environment and some property values so the example will run. In order for the examples to run, you must:

- have the B2B Engine running.
- have your CLASSPATH set properly.
- set your connection properties.

IMPORTANT: These examples will fail if the B2B Engine is not running.

SETTING YOUR CLASSPATH

To configure your environment, you must include the External API jar file in your CLASSPATH environment variable. For the examples, the easiest way to do this is to run the example applications from the Diagnostic Shell. For your applications, you should add the External API jar file to your CLASSPATH. You can find the External API jar file in:

On this platform	The file is
Windows	Websphere\PAM\Alliance\externalapi.jar

SETTING YOUR CONNECTION PROPERTIES

The property file named Example.properties contains property information that is used by all the example programs. You must change the information in this file to match your installation. The properties file is a simple text file which you can modify in any text editor.

IMPORTANT: The examples will not run unless you change this property information.

You can find the Example.properties file in:

On this platform	The file is
Windows	Partner*nnn*\com\extricity\api\example\Example.properties

In the Example.properties file, you must set your connection properties. You specified all of these values when you installed the B2B Engine.

Set this value	To
server.host	The name of the machine running your B2B Engine.
server.port	The port that your B2B Engine is listening on.

Set this value	To
server.user	The name of a user who is already set up in the B2B Engine. This user must have permissions for the operations the code will perform.
server.password	The password for the user.

Importing Java classes in the examples

Each example imports Java classes from the Java libraries and the External API for services it will use.

This class	Does this
com.extricity.api.example com.extricity.api.example.APILibrary	Includes code common to all the examples.
com.extricity.api.example.ExampleProperties	Gets property values.
com.extricity.api.rmi.RMIClientSession com.extricity.api.Session	Creates a connection to the B2B Engine.
com.extricity.process.api.*	Manages processes.
com.extricity.document.api.BusinessObjectType com.extricity.document.api.BusinessObjectTypeRef	Manages BusinessObjectTypes and type refs.
com.extricity.document.api.DocumentService	Manages business object schemas and BusinessObjectTypes.
com.extricity.document.api.DocumentServiceFactory	Provides access to the DocumentService.
com.extricity.document.api.ElementDefSet	Represents a set of element definitions that can be used to define a BusinessObjectType.
com.extricity.document.api.ElementDefSetRef	Represents an ElementDefSet.
com.extricity.adapter.api.*	Manages adapters.

This class	Does this
com.extricity.api.SessionException com.extricity.api.SystemException, com.extricity.api.SystemSecurityException com.extricity.api.RefException com.extricity.api.QueryException com.extricity.api.ServiceException com.extricity.api.SystemIOException	Contains External API exception classes.
java.io.StringWriter java.text.SimpleDateFormat java.util.Collection java.util.Iterator java.util.Date java.io.StringReader java.io.BufferedInputStream java.io.FileInputStream java.io.FileNotFoundException java.io.IOException, java.io.InputStream	Contains Java library classes.

Classes that are common to the examples

There are some additional classes that are common to all the examples in the main examples folder.

On this platform	The files are
Windows	Partner*nnn*\com\extricity\api\example\APILibrary.java
	Partner*nnn*\com\extricity\api\example\ ExampleProperties.java

2

Connecting to the B2B Engine

Every application that uses the External API will connect to the B2B Engine. Read this chapter for a demonstration of how to use the External API in your application to connect and log on to the B2B Engine.

This chapter includes these sections:

- *What this example does* on page 10.
- *About the source code for this example* on page 10.
- *Running this example* on page 10.
- *Connecting and logging in to the B2B Engine* on page 11.
- *Getting property values* on page 13.
- *Extending this example* on page 14.

What this example does

This very simple example connects to the B2B Engine and logs on. It then logs off, closing the connection gracefully.

About the source code for this example

The complete source code for this example is under your Partner folder:

On this platform	The file is
Windows	Partner*nnn*\com\extricity\api\example\ SessionExample.java

NOTE: This document focuses on the most important parts of each example. Some lines of code are not covered. The source code also includes explanatory comments.

Running this example

Running this example is a good way to check that your system is set up correctly to use the External API.

To run this example:

1 Make sure that you've satisfied the requirements listed in *Before you run the examples* on page 5.

2 From the Diagnostic Shell, type this at the command line:

java com.extricity.api.example.SessionExample

Troubleshooting

If you tried to run the example and it was not successful, there are a few things to check:

- In Example.properties, are the property values for the host and port correct?

- In Example.properties, are the property values for the user and password correct? Does that user already exist in the B2B Engine?

- Are both the local and B2B Engine machines accessible on the network?

- Is the External API jar file correctly set in the CLASSPATH?
- Is the B2B Engine running?

Connecting and logging in to the B2B Engine

After importing Java packages, the example tries to connect to the B2B Engine and log in.

Connecting to the B2B Engine consists of two steps:

Step 1 Get the connection properties information.

Step 2 Use that information to attempt to make a connection to the B2B Engine.

Getting the connection properties

The first step in connecting to the B2B Engine is to get the connection information. The ExampleProperties class reads in the properties file and gets the property values. The lookupValue method in that class does the lookup work. The ExampleProperties class is in the com.extricity.api.example.ExampleProperties package, which was imported in the beginning of the example.

First, create a new instance of the ExampleProperties class:

```
ExampleProperties ep = new ExampleProperties();
```

Inside a try block, get the necessary property values and use them to create a connection and log in to the B2B Engine. The values are server.host, server.port, server.user, and server.password. For more information about setting these values, see *Before you run the examples* on page 5. For more on how the ExampleProperties class parses the properties file, see *Getting property values* on page 13.

The example declares the variable session and attempts to look up the connection information and connect to the B2B Engine.

```
RMIClientSession session = null;
session = new RMIClientSession(ep.lookupValue("server.host"),
        Integer.parseInt(ep.lookupValue("server.port")));
```

Logging in

Once the host and port are known, look up the user name and password, and attempt to log in:

```
session.login(ep.lookupValue("server.user"),
    ep.lookupValue("server.password"));
```

If this is successful, you have connected and logged in to your B2B Engine.

Next, print out some notifications. The notifications use the isLoggedIn method of the Session interface. isLoggedIn returns a boolean: true if the user is logged in to the B2B Engine, false if not.

```
System.out.println(ep.lookupValue("server.user") +
        "is logged into session: " +
        session.isLoggedIn());
```

Logging out

When your application has finished with the B2B Engine, you should log out and close the connection gracefully. The logout method of the RMIClientSession class does this.

```
session.logout();
```

To check that this worked properly, the example then prints a debugging message, again using the isLoggedIn method.

```
System.out.println("\nLogging out of session\n");
System.out.println(ep.lookupValue("server.user") +
        "is logged into session: " +
        session.isLoggedIn());
```

After the try block, catch the exceptions. This example just catches them and prints a stack trace. In a real application, you'll catch the exceptions and handle the errors properly.

That's it. You've connected to the B2B Engine, logged in, and logged out.

Getting property values

For the purposes of this example, you can treat the property values as a black box. However, property values are a good way to avoid hard-coding values into your applications. The ExampleProperties class parses the properties file and reads the specified property value. You can find the complete source for this class in:

On this platform	The file is
Windows	Partner*nnn*\com\extricity\api\example\ExampleProperties.java

The class declares the variable bundle as a private variable:

```
private ResourceBundle bundle;
```

Next, the name of the property file is hard-coded. A less trivial application would take the name of the property file as an argument instead.

```
public static final String example_properties =
     "com.extricity.api.example.Example";
```

Next, load the property file:

```
bundle = ResourceBundle.getBundle(example_properties);
```

Then, get the key or return null if the property file is not loaded.

```
public String lookupValue(String key) {
  if (bundle == null) {
     return(null);
  } else {
     return(bundle.getString(key));
  }
}
```

Extending this example

You could easily extend this simple example to connect to two B2B Engines at the same time. This would be useful, for example, in migrating a process from a test system to a production system. To automate migrating the process from test to production, you could write an application that connects to the test system and exports the process. Your application could then connect to the production system and import it.

To extend this example to connect to two B2B Engines:

1 In the Example.properties file, add a new set of properties and values for the second B2B Engine, say servertest.port, servertest.host, servertest.user and servertest.password.

2 In the ExampleSession.java file, create a new session variable, for example, session_2.

```
RMIClientSession session_2 = null;
```

3 Look up the new B2B Engine information and connect to it:

```
session_2 = new
RMIClientSession(ep.lookupValue("servertest.host"),
    Integer.parseInt(ep.lookupValue("servertest.port")));
session_2.login(ep.lookupValue("servertest.user"),
    ep.lookupValue("servertest.password"));
```

You now have two concurrent connections, each to a different B2B Engine.

You could also extend this example to get the connection properties from a command line, from another application or from user input.

3

AUDITING PROCESSES

Applications using the External API can automate auditing of processes. Read this chapter for a demonstration of how to get information about processes in the B2B Engine.

This chapter includes these sections:

What this example does

This example application displays information about processes and their status. If you give it a process ID as an argument, it will retrieve more detailed information about that process. If you give it no argument, it will display information about all processes.

This example demonstrates:

- how to get a reference to a particular process.
- how to get a list of all processes that match specified criteria.
- how to access different pieces of process data.

About the source code for this example

The complete source code for this example is under your Partner folder.

On this platform	The file is
Windows	Partner*nnn*\com\extricity\api\example\process\ AuditApp.java

Note: This document focuses on the most important parts of each example. Some lines of code are not covered. The source code also includes explanatory comments.

Running this example

Running this example is a good way to see how the External API gets information about processes.

Before you run this example

Step 1 Make sure that you've satisfied the requirements listed in *Before you run the examples* on page 5.

Step 2 To see this most clearly, you must have some processes set up, preferably in various different states before you run this example. If you have no processes when you run this example, you won't get any useful output.

RUNNING THIS EXAMPLE

There are two ways to run this example, with or without a PublicProcessRef.

To run this example without a PublicProcessRef:

▶ From the Diagnostic Shell, type this at the command line:

```
java com.extricity.api.example.process.AuditApp
```

This gives you a short summary of all processes in the system. The short summary will include a PublicProcessRef for each process. This PublicProcessRef is a string that is a unique identifier for the public process. To get more detailed information on a given process, run this example with the PublicProcessRef for that process as an argument:

To run this example with a PublicProcessRef:

▶ java com.extricity.api.example.process.AuditApp *PublicProcessRef*

This gives you detailed information about the public process the PublicProcessRef refers to.

NOTE: If your process name has spaces in it, make sure to add quotes around the PublicProcessRef string.

TROUBLESHOOTING

If you tried to run the example and it was not successful, there are a few things to check:

- Do you have any processes running? If you don't, the example will complete successfully, but you won't get any useful information.
- Do you have the proper permissions for viewing the audit file?
- Are the property values for the host and port correct?
- Are the property values for the user and password correct? Does that user already exist in the B2B Engine?
- Are both the local and B2B Engine machines accessible on the network?
- Is the External API jar file correctly set in the CLASSPATH?
- Is the B2B Engine running?

Getting the ProcessService

The application starts by doing some argument checking and then it connects to the server and logs in.

Once you have a connection and are logged in, get the ProcessService. The ProcessService allows you to manage processes and events in the B2B Engine. Here, you use the ProcessService to get information about processes. Other ProcessService functions are demonstrated in *Starting processes* on page 23 and *Using events with processes* on page 31.

The ProcessServiceFactory gives you the ProcessService for this session. Each session has only one instance of any given service.

```
ProcessService process_service = null;
process_service = ProcessServiceFactory.getService(session);
```

Getting information about all public processes

If you didn't give a PublicProcessRef as a command-line argument, the example queries for information about all processes. To query for information, you must first construct a Query object.

First, build a new query object and do some date arithmetic. The date arithmetic sets the start_date to be 24 hours ago.

```
query = new PublicProcessQuery();
Date start_time = new Date(new Date().getTime() -1000*60*60*24);
```

Next, set the fields in the query. These are matched against the processes and information on all matching processes is returned. In this case, it queries for all processes that have begun in the last 24 hours. You could, of course, set this time to any interval you like.

```
query.setStartTime(start_time);
query.setEndTime(new Date());
```

You can set other fields in the query as well.

Using PublicProcessAgents

The result of the query is a Collection of PublicProcessAgents. A PublicProcessAgent represents a public process. You use the PublicProcessAgent to view and control the execution of its public process instance. Each PublicProcessAgent is associated with one and only one public process instance.

After printing out the label information, do the query and get any matching PublicProcessAgents. These PublicProcessAgents contain the detailed information about any processes that matched the query.

```
public_process_agents =
process_service.getPublicProcessAgents(query);
```

Print out the information in the PublicProcessAgents.

```
APILibrary.printProcessAgent(public_process_agents, sdf);
```

sdf refers to the SimpleDateFormat. Here it indicates that you want PublicProcessAgents for all public processes that were run in the last 24 hours.

After the try block, catch the exceptions, and then close the connection gracefully. This example just catches the exceptions and prints a stack trace. In your applications, you'll catch the exceptions and handle errors.

Getting information on particular public processes

After you have gotten information about all processes, you can use the PublicProcessRefs returned there to get information about a particular public process. If you gave a PublicProcessRef as an argument to this AuditApp application, it parses that string into a PublicProcessRef object.

```
public_process_ref = PublicProcessRef.get(args[0]);
```

With that PublicProcessRef object, you can get the PublicProcessAgent for the public process referred to by the PublicProcessRef.

```
public_process_agent =
     process_service.getPublicProcessAgent(public_process_ref);
```

From this line of code, you can see that the ProcessService retrieves the PublicProcessAgent. This PublicProcessAgent is a snapshot of the public process instance at the time the PublicProcessAgent is retrieved from the ProcessService. As the public process instance executes, the information contained in the PublicProcessAgent will become out of sync with the process instance. To update the information in the PublicProcessAgent, call the refresh method. Once the public process instance has completed its execution, the information will no longer change.

TIP: Refresh takes a parameter that is a constant. This constant indicates the information to be refreshed. This method uses a lot of resources, so where performance is a consideration, only refresh the information you need.

If a PublicProcessAgent was successfully retrieved for the PublicProcessRef, print out the process agent.

```
if (public_process_agent != null) {
  APILibrary.printProcessAgentFull(public_process_agent,
        process_service,sdf);
```

In the APILibrary, printPublicProcessAgent simply iterates through the PublicProcessAgent, printing the PublicProcessRef, the start time, the end time, and the state. Then it calls printProcessAgentSummary to print out a short summary of each public process instance:

```
public static void printProcessAgent(Collection agents,
            SimpleDateFormat date_format) {
  PublicProcessAgent agent;
  Iterator agent_itr = agents.iterator();
  System.out.println("Public Process Ref " + "\t\t\t\t\t\t" +
          "Start Time (MM/DD/YY)" + "\t" +
          "End Time (MM/DD/YY)" + "\t" +
          "State");
  while(agent_itr.hasNext()) {
      agent = (PublicProcessAgent) agent_itr.next();
      printProcessAgentSummary(agent, date_format);
  }
}
```

If nothing was found matching that PublicProcessRef, the PublicProcessAgent is null.

After it's done, catch the exceptions, and close the connection gracefully. This example just catches the exceptions and prints a stack trace. In your non-trivial applications, you'll catch the exceptions and handle the errors.

Extending this example

You could extend this example application into a Web site that serves the same information.

You could also extend this example to do something based on the results of the query.

4

STARTING PROCESSES

One of the most common uses for the External API is to start processes. Read this chapter to see how to find process instances, create new process instances, set input, and start processes with the External API.

This chapter includes these sections:

What this example does

This example demonstrates:

- how to find a process type based on a process name.
- how to create a new instance of a process type.
- how to set variant and business object inputs to a process.
- how to start a process with an external process ID.
- how to get the results of that process.

About the source code for this example

The complete source code for this example is under your Partner folder.

On this platform	The file is
Windows	Partner*nnn*\com\extricity\api\example\process\ ProcessStartApp.java

Note: This document focuses on the most important parts of each example. Some lines of code are not covered. The source code also includes explanatory comments.

Running this example

Running this example is a good way to see more details about how the External API works with processes.

Before you run this example

Step 1 Make sure you have satisfied the requirements in *Before you run the examples* on page 5.

Step 2 Set up a process for this application to start.

Setting up a process for this application to start

You must have a process that can be started by this example application. See the *Partner Agreement Manager User's Guide* for more information on creating processes, business objects, and partner profiles.

To set up a process for this example to start:

1 Set up your local partner profile, add a listener, and add your certificates.

   NOTE: If your process involves more than one partner, you must exchange profiles with your partners.

2 Create a public process called "Example Process."

3 Create an input variant called "Purchase_Order_Number."

4 Create an input business object called "Input_File" of the type File.

5 Create an output variant called "Purchase_Ack_Number."

6 Create an output business object called "Output_File" of type File.

7 Implement the private process so the output variant and business object are populated.

   The example application queries these context variables.

8 Print out the input variant and business object.

   This is populated by the example application.

9 Install the process for testing.

   IMPORTANT: If you change the name of the public process or any of the input or output variants, you must also change the corresponding values in the Example.properties file.

Running this example

There are 2 ways to run this example, without an external ID or with one.

To run this example without an external ID:

▶ From the Diagnostic Shell, type this at the command line:

java com.extricity.api.example.process.ProcessStartApp

This starts the process named by the property process.name with the inputs named by process.input_variant and process.input_bo. These inputs have the values process.input_variant_value and process.input_bo_value, respectively.

To run this example with an external ID:

▶ From the Diagnostic Shell, type this at the command line:

java com.extricity.api.example.process.ProcessStartApp *External_ID*

This starts the process named by the property process.name with the inputs named by process.input_variant and process.input_bo. These inputs have the values process.input_variant_value and process.input_bo_value, respectively. These are the values set in the Example.properties file.Any argument you give on the command line will be used as a unique external ID to the process. You can then use this external ID to query the process.

An external ID can be any unique identifier, such as a unique purchase order number or a unique material number. You can use any unique number that make sense in the context of the process and your organization.

TROUBLESHOOTING

If you tried to run the example and it was not successful, there are a few things to check:

- Do you have the required process defined? If not, the example will fail.
- Does your process, along with its inputs and outputs, have the same name as in the Example.properties file?
- Are the property values for the host and port correct?
- Are the property values for the user and password correct? Does that user already exist in the B2B Engine?
- Are both the local and B2B Engine machines accessible on the network?
- Is the External API jar file correctly set in the CLASSPATH?
- Is the B2B Engine running?

GETTING PROCESS INFORMATION

First connect to the B2B Engine and get the ProcessService. For more information about connecting to the B2B Engine, see *Connecting to the B2B Engine* on page 9. For more information about getting the ProcessService and what it does, see *Getting the ProcessService* on page 18.

Next, get some basic information about the process. Look up the name of the public process from the Example.properties file. From the name, get the PublicProcessType. From the PublicProcessType, get the PublicProcessTypeRef.

IMPORTANT: To get the process name, the example looks up the value of the process.name property in the Example.properties file. If this property value and your process name do not exactly correspond, the example will fail.

```
process_type = APILibrary.getPublicProcessType(process_service,
  ep.lookupValue("process.name"));
process_type_ref = process_type.getPublicProcessTypeRef();
```

NOTE: The APILibrary method getPublicProcessType, sets a Query object with the name of the public process and runs the query.

GETTING THE PUBLICPROCESSAGENT

With the PublicProcessTypeRef you just got, you can get a PublicProcessAgent, which allows you to control the process. For more information about PublicProcessAgents, see *Using PublicProcessAgents* on page 19. To get the public process agent, call the ProcessService method newPublicProcess.

```
public_process_agent =
    process_service.newPublicProcess(process_type_ref,
    ExecutionMode.AUTO_SELECT, null);
```

If you provided an external ID in the argument list for this example application, it will use that ID.

```
public_process_agent =
    process_service.newPublicProcess(process_type_ref,
    ExecutionMode.AUTO_SELECT, args[0]);
```

If you didn't provide an external ID as an argument, the PublicProcessRef can be used as an ID for referencing the process later.

NOTE: By setting the execution mode of the public process to AUTO_SELECT, the process is started in Production mode if you installed it for production and Test mode if you installed it for testing.

SETTING THE INPUTS

Now that you have the PublicProcessAgent, you must set the inputs before starting the process.

SETTING THE INPUT VARIANTS

First, use the PublicProcessAgent to determine the inputs the process takes. Look up the input_variant property in Example.properties and check that name in the execution input. If it matches, set the input variant value with setValue.

```
input = public_process_agent.getExecutionInput();
vcv =
      input.getVariantContextVariable(ep.lookupValue("process."+
                  "input_variant"));
if (vcv != null) {
  vcv.setValue(ep.lookupValue("process.input_variant_value"));
} else {
  System.out.println("Input variant " +
      ep.lookupValue("process.input_variant") +
      " was not found in the process type. Not " +
      "setting context variants");
}
```

If the name does not match the input variant required by the process, print out an error message.

SETTING THE INPUT BUSINESS OBJECT

Getting and setting the data input business objects works in exactly the same way as setting the data of the input variants. To do this, look up the name of the input business object from the PublicProcessAgent. You already have the execution inputs from the getExecutionInput call in *Setting the input variants*. If the name of the required input business object and property match, set the value from the process.input_bo_value in Example.properties. This value is an XML string that contains the DOCTYPE and the business object instance data. If the name does not match the input variable or business object required by the process, print out an error message.

```
bocv =
      input.getBusinessObjectContextVariable(ep.lookupValue
              ("process."+ "input_bo"));
if (bocv != null) {
```

```
      bocv.setValueFrom(new StringReader(ep.lookupValue("process."+
            "input_bo_value")));
} else {
   System.out.println("Input BO " +
            ep.lookupValue("process.input_bo") +
            " was not found in the process type. " +
            "Not setting context variants");
}
```

NOTE: Note that the lookup value is retrieving the XML instance of the business object from the property file, but you could also get it from a file or some other InputStream.

STARTING THE PROCESS INSTANCE

Now that you have the process agent and the process inputs are set, you can start the process.

```
public_process_agent.startProcess();
```

CHECKING ON THE STATE OF THE PROCESS

After you've started the process, the state of the process is IN_PROGRESS. Once you've started the process, wait until the process has changed state to get the results. This does not necessary mean that the process has finished, it could also be aborted, committed, or suspended.

Check the state of the process. This PublicProcessAgent contains a snapshot of the current process state. If the process state is IN_PROGRESS, wait 5 seconds before checking again.

```
while (public_process_agent.getProcessExecutionState() ==
      ProcessExecutionState.IN_PROGRESS) {
   try {
      Thread.sleep(5000); //wait 5 seconds before checking again
```

After waiting, refresh the information in the snapshot. Since you just need the state refreshed, you can refresh just that state information. Using this method to refresh just what you need is faster than refreshing everything in the PublicProcessAgent.

If the process instance controlled by this public process agent has not been started or does not exist, print out a stack trace and return. In an application in a production environment, error handling and recovery would happen gracefully here.

```
public_process_agent.refresh(PublicProcessAgent.REFRESH_STATE);
  } catch (Exception e) {
     System.out.println("Couldn't refresh public process " +
             "agent's state.");
     e.printStackTrace();
     return;
  }
}
```

Getting the outputs from the process

Now that the process is no longer in progress, get the full details on the process and print them out. To get the full details, you need to refresh everything to be able to print it all out.

```
public_process_output.refresh.REFRESH_ALL);
APILibrary.printProcessAgentFull(public_process_agent,
     process_service,
     sdf);
```

This gives you the details on the what the process did.

5

USING EVENTS WITH PROCESSES

Read this chapter for information on using the External API to start processes using events.

This chapter includes these sections:

What this example does

This example application shows how to start processes by creating EventTypes and Event instances. If your EventType has not been created, it creates one and registers the process with the EventType.

This example demonstrates:

- how to find an EventType.
- how to create a new EventType with business object data.
- how to register a process to an EventType.
- how to create a new Event.
- how to find out what happened to an Event after it was posted.

About the source code for this example

The complete source code for this example is under your Partner folder.

On this platform	The file is
Windows	Partner*nnn*\com\extricity\api\example\process\ProcessEventApp.java

Note: This document focuses on the most important parts of each example. Some lines of code are not covered. The source code also includes explanatory comments.

Running this example

Running this example is a good way to see more details about how the External API works with Events and processes.

Before you run this example

Step 1 Make sure you have satisfied the requirements in *Before you run the examples* on page 5.

STEP 2 Set up a process to be started by this application. For more information about how to set up the process, see *Setting up a process for this application to start* on page 24. If you already set up this example process to run the StartApp example, you don't need to do it again.

Set the values of the event properties in the Example.properties file.:

Set this value	To
event.type	The EventType to use to start the process.
event.partner_id	The Partner ID of your partner in this process.
event.partner_name	The name of your partner in this process.
event.bo_root_name	The name of the root element of this business object.
event.bo_eds_name	The name of the business object element definition set.

Running this example

There are 2 ways to run this example, without an external ID or with one.

To run this example without an external ID:

▶ From the Diagnostic Shell, type this at the command line:

java com.extricity.api.example.process.ProcessEventApp

This starts the process named by the property process.name with the inputs named by process.input_variant and process.input_bo. These inputs have the values process.input_variant_value and process.input_bo_value, respectively. These are the values given in the Example.properties file.

To run this example with an external ID:

▶ From the Diagnostic shell, type this at the command line:

java com.extricity.api.example.process.ProcesEventApp *External_ID*

This starts the process named by the property process.name with the inputs named by process.input_variant and process.input_bo. These inputs have the values process.input_variant_value and process.input_bo_value, respectively. Any argument you give on the command line is used as an unique external ID to the process. The process can then be queried using this external ID.

An external ID can be any unique identifier, such as a unique purchase order number or a unique material number. You can use any unique number that make sense in the context of the process and your organization.

NOTE: If you don't provide an external ID, you must use the EventRef to retrieve the EventExecutionSummary after the Event has been posted.

TROUBLESHOOTING

If you tried to run the example and it was not successful, there are a few things to check:

- Do you have any processes defined? You must have already defined the processes you will register with the EventType. If you don't, the example will fail.
- Does your process, along with its inputs and outputs, have the same name as in the Example.properties file?
- Are the property values for the host and port correct?
- Are the property values for the user and password correct? Does that user already exist in the B2B Engine?
- Are both the local and B2B Engine machines accessible on the network?
- Is the External API jar file correctly set in the CLASSPATH?
- Is the B2B Engine running?

SETTING UP THE CONNECTION AND SERVICE

First, connect to the B2B Engine and get the ProcessService. For more information about connecting to the B2B Engine, see *Connecting to the B2B Engine* on page 9. For more information about getting the ProcessService and what it does, see *Getting the ProcessService* on page 18.

GETTING THE EVENT TYPE

Since you'll be starting this process with an event, you need to get the EventType. To do this, use an EventTypeQuery object. First, create a new EventTypeQuery object.

```
query = new EventTypeQuery();
```

Set the name in the EventTypeQuery. Get this name from the event.type property in the Example.properties file. The default name is New Process Event.

```
query.setName(ep.lookupValue("event.type"));
```

Do the query with the EventTypeQuery.

```
event_types = process_service.getEventTypes(query);
```

The query will return event_types, which is a Collection. However, for the purposes of this brief example, assume that this is a unique event and do not iterate past the first value returned.

```
itr = event_types.iterator();
if(itr.hasNext()) {
  event_type = (EventType) itr.next();
} else {
```

If the EventType was not found, create the Event with business object data. To do this, look up the event.bo* properties from the Example.properties file. If they're found, print out a diagnostic listing the root element name.

```
BusinessObjectType bo_type;
bo_type = APILibrary.getBOType(session,
        ep.lookupValue("event.bo_partner_id"),
        ep.lookupValue("event.bo_partner_name"),
        ep.lookupValue("event.bo_root_name"),
        ep.lookupValue("event.bo_eds_name"),
        null, null);
if (bo_type != null) {
    System.out.println("found bo_type: " +
            bo_type.getRootElementName());
```

If you've found the business object, get the BusinessObjectTypeRef, and use that, along with the EventType from the Example.properties file to generate a new EventType.

```
    event_type =
```

```
            process_service.newEventType(ep.lookupValue
                ("event.type"),
            EventType.NO_TIMEOUT,
            bo_type.getBusinessObjectTypeRef());
        } else {
```

If you couldn't find a business object to create the event, print a message and create the Event with variant data.

```
System.out.println("Couldn't find a BO to create the event " +
        "with, creating an event with variant "+
        "data");
event_type = process_service.newEventType(ep.lookupValue
            ("event.type"),
         EventType.NO_TIMEOUT);
```

REGISTERING A PROCESS FOR THIS EVENT TYPE

Once you have the EventType, the next step is to register a process for this EventType. Look up the process name from the Example.properties file and get the type of that process.

```
PublicProcessType process =
        APILibrary.getPublicProcessType(process_service,
        ep.lookupValue("process.name"));
```

If you created the EventType with business object data, register the process to receive business object input when it is triggered by this EventType. Otherwise, register it with a variant input. Then, save the EventType.

```
if (bo_type != null) {
  event_type.registerProcess(process.getPublicProcessTypeRef(),
            ep.lookupValue("process.input_bo"));
} else {
  event_type.registerProcess(process.getPublicProcessTypeRef(),
            ep.lookupValue("process."+
            "input_variant"));
}
event_type.save();
```

CREATING A NEW EVENT

After you've registered the process with the event type, create a new Event. If you specified an external event ID on the command line, use it. Otherwise, create an Event with no ID.

```
if (args.length > 0) {
  event = process_service.newEvent(event_type.getEventTypeRef(),
        ExecutionMode.TEST,
        args[0]);
} else {
  event = process_service.newEvent(event_type.getEventTypeRef(),
        ExecutionMode.TEST,
        null);
}
```

SETTING THE DATA FOR THE EVENT AND POST

The last step before posting the Event is to set the data in the event. This data is passed as an input to each of the registered processes. If you are using a business object input for the Event, use the values set in the input_bo_data property in the Example.properties file. Print out a diagnostic with the event data. Note that the data is gotten as a string and cast to BusinessObjectEventData.

```
APILibrary.setEventData(event,
        ep.lookupValue("process.input_bo_value"));
System.out.println("event data: " + ((BusinessObjectEventData)
            event.getEventData()).getValueAsString());
```

If there is no business object input data, the setEventData method sets variant data for the event instead.

Post the Event.

```
event.post();
```

GETTING THE EXECUTION SUMMARY

After the Event is posted, get the event execution summary. If you gave an event ID in the command line, use that to get the summary information.

```
if (args.length > 0) {
  event_summary = process_service.
        getEventExecutionSummary(event_type.getEventTypeRef(),
        args[0]);
} else {
  event_summary = process_service.
        getEventExecutionSummary(event.getEventRef());
}
```

Print out the event summary information.

```
System.out.println("Creation Time:       " +
             event_summary.getCreationTime());
System.out.println("Event Instance ID:   " +
             event_summary.getEventInstanceID());
System.out.println("Event Ref:           " +
             event_summary.getEventRef());
System.out.println("Event State:         " +
             event_summary.getEventState().toString());
System.out.println("Event State Details: " +
             event_summary.getEventStateDetails());
System.out.println("Execution Mode:      " +
             event_summary.getExecutionMode().toString());
```

Print out a list of processes that were started.

```
System.out.println("Processes that were started:");
started_processes =
     event_summary.getPublicProcessInitiationSummaries();
itr = started_processes.iterator();
while(itr.hasNext()){
  System.out.println((PublicProcessInitiationSummary)
             itr.next());
}
```

Catch the exceptions and handle the errors.

6

Using the Document API

Read this chapter for an illustration of how to create a new ElementDefinitionSet and BusinessObjectType from the ExampleXML.dtd file.

This chapter includes these sections:

What this example does

This example demonstrates:

- how to import and freeze an ElementDefinitionSet.
- how to create a business object from the ElementDefinitionSet.

About the source code for this example

The complete source code for this example is under your Partner folder:

On this platform	The file is
Windows	Partner*nnn*\com\extricity\api\example\document\ DocumentApp.java

Note: This document focuses on the most important parts of each example. Some lines of code are not covered. The source code also includes explanatory comments.

Running this example

Running this example is a good way to see how to manipulate business objects in your application.

To run this example:

Step 1 Make sure that you've satisfied the requirements listed in *Before you run the examples* on page 5.

Step 2 From the Diagnostic Shell, type this at the command line:

```
java com.extricity.api.example.document.DocumentApp
```

Setting up the connection and services

First, connect to the B2B Engine, and get the document and partner services. For more information about connecting to the B2B Engine, see *Connecting to the B2B Engine* on page 9. For more information about services in general, see *Using services* on page 3, for more information about getting a service, see *Getting the ProcessService* on page 18.

IMPORTING THE ELEMENT DEFINITION SET

Look up the name of the file containing the business object schema in XML format and open up a FileInputStream for that file. Get the local partner ref. Then, using the document service method importPartnerElementDefSet import the element definition set. Using the document service method getElementDefSet get the imported element definition set. Mark it as frozen. Since it is owned by the local partner, the ElementDefSet is marked as frozen and will not be editable. You do not need to save it.

```
ElementDefSetRef eds_ref = null;
ElementDefSet eds = null;
FileInputStream fis;

fis = new FileInputStream(ep.lookupValue("bo.file_name"));
local_partner_ref =
        partner_service.getLocalPartner().getPartnerRef();
eds_ref = document_service.importPartnerElementDefSet(fis,
        ep.lookupValue("bo.eds_name"),
        ep.lookupValue("bo.eds_external_id")
        local_partner_ref);
eds = document_service.getElementDefSet(eds_ref);
eds.markAsFrozen();
```

This is not the only way to import an ElementDefinitionSet. The ways to import an ElementDefinitionSet are:

- Use the importPartnerElementDefSet method to import an XML DTD into a particular partner. This partner can be either a local partner or another partner. The DTD can be any DTD. If the local partner is used, you don't need to freeze the ElementDefinitionSet.

- Use the importStandardElementDefSet method to import a full DTD into the Standards folder in Partner Agreement Manager. Standards refers to a standard, such as RosettaNet.

- Use the importElementDefSet method to import a business object that was exported from Partner Agreement Manager.

IMPORTANT: Before importing a DTD using the External API, you must resolve all external entities in that DTD.

Resolving external entities in a DTD

If you attempt to import a DTD that has external entities, the import will fail.

There is a utility provided with the B2B Engine that will do that for you. The utility is com.extricity.document.apps.dtd.InlineExternalEntities.

To resolve external entities in a DTD:

1 On the same machine as B2B Engine, make a directory or directory tree containing the DTD and its external entities.

2 In the Diagnostic Shell, got to the directory containing the main DTD file and type this at the command line:

com.extricity.document.apps.dtd.InlineExternalEntities *mainDTDfile newDTDfile*

mainDTDfile is the name of the DTD file that contains the external entities. *newDTDfile* is the name of the file that will contain the DTD file with all external entities resolved. You can then import this new DTD file into the B2B Engine.

Creating a business object from the element definition set

To create a business object from the ElementDefinitionSet you just imported and froze, use the newBusinessObjectType method of the DocumentService.

```
BusinessObjectType bo_type = null;
bo_type = document_service.newBusinessObjectType(eds_ref,
                ep.lookupValue("bo.root_name"));
bo_type.save();
```

Catch the exceptions and handle any errors.

7

Using the External API with adapters

Read this chapter for information on how to use the External API to manage adapters.

This chapter includes these sections:

What this example does

This example demonstrates:

- how to import an adapter type.
- how to import an adapter implementation.
- how to import an adapter instance.
- how to start an adapter.
- how to print an adapter's status.

This type of application is useful for moving an adapter from a test system into a production one.

About the source code for this example

The complete source code for this example is under your Partner folder.

On this platform:	the file is:
Windows	Partner*nnn*\com\extricity\api\example\adapter\ AdapterApp.java

Note: This document focuses on the most important parts of each example. Some lines of code are not covered. The source code also includes explanatory comments.

Running this example

Running this example is a good way to see more details about how the External API works with adapters.

Important: Make sure you have satisfied the requirements in *Before you run the examples* on page 5.

Running this example

There are three XML files in the adapter examples folder, one each for the adapter type, implementation, and instance. There is also ExampleAdapter.java, which is the source code for the adapter imported by this example. A look at that code will show you that this adapter doesn't actually do anything. This example shows how to import those files and start the adapter.

To run this example:

▶ From the Diagnostic Shell type this at the command line:

```
java com.extricity.api.example.adapter.AdapterApp
```

This runs the Adapter Application, which imports the adapter type, implementation, and instance specified in the Example.properties file. If any of these objects already exist, this application throws an exception.

Troubleshooting

If you tried to run the example and it was not successful, there are a few things to check:

- Do you have an adapter type named Example Adapter, an adapter implementation named Example Adapter Java Implementation or an adapter instance named Example Adapter Instance? If so, this example will fail, as there is a conflict between your names and the ones in the example adapter.
- Are the property values for the host and port correct?
- Are the property values for the user and password correct? Does that user already exist in the B2B Engine?
- Are both the local and B2B Engine machines accessible on the network?
- Is the External API jar file correctly set in the CLASSPATH?
- Is the B2B Engine running?

Setting up the connection and service

First, connect to the B2B Engine and get the adapter service. For more information about connecting to the B2B Engine, see *Connecting to the B2B Engine* on page 9. For more information about getting services, see *Getting the ProcessService* on page 18.

Note: The adapter type, implementation, and instance are all files in the file system. They have been exported from the B2B engine. For this example, the adapter has already been written and this export has already been done for you.

Importing the adapter type and implementation

To import the adapter type, look up the adapter type file name from Example.properties and open a FileInputStream to read the input file. Then call the importAdapterType method of the AdapterService.

```
FileInputStream fis = null;
fis = new FileInputStream(ep.lookupValue("adapter.type_file"));
adapter_service.importAdapterType(fis);
```

You import the adapter implementation in exactly the same way.

```
fis = new FileInputStream(ep.lookupValue("adapter.impl_file"));
adapter_service.importAdapterImplementation(fis);
```

Importing the adapter instance

Importing the adapter instance is similar to importing an adapter implementation, but with one difference. After the FileImputStream is created, the instance is imported and the return value of importAdapter, the AdapterRef, is kept.

```
AdapterRef adapter = null;
fis = new FileInputStream(ep.lookupValue("adapter.inst_file"));
adapter = adapter_service.importAdapter(fis);
```

Starting the Adapter

Using the AdapterRef, get the AdapterAgent. Analogous to the PublicProcessAgent used in *Starting processes* on page 23 and *Using events with processes* on page 31, the AdapterAgent is used to control the adapter. Use the AdapterRef, adapter to get the AdapterAgent. Then, use the AdapterAgent to start the adapter.

```
AdapterAgent adapter_agent = null;
adapter_agent = adapter_service.getAdapterAgent(adapter);
adapter_agent.startAdapter();
```

NOTE: The adapter will only start if the adapter class file, pointed to by the adapter implementation, can be found. In the example, the class file is called ExampleAdapter.class. This class file is included in the distribution.

Since this AdapterAgent is a snapshot of information about the adapter at a particular point in time, get it again to refresh the information, since starting the adapter probably caused some information to change.

```
adapter_agent = adapter_service.getAdapterAgent(adapter);
```

Printing out the adapter status

With the refreshed information, print out the status of the adapter.

```
System.out.println(adapter_agent.getName() + " status: " +
            adapter_agent.getStatus());
```

This status indicates whether the adapter is running.

As always, catch the exceptions and do proper error handling.

Extending this example

You could extend this example to detect a suspended adapter. You could then use the application to do a preliminary diagnosis of the problem, perhaps attempt to fix it, and re-start the adapter.

A

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

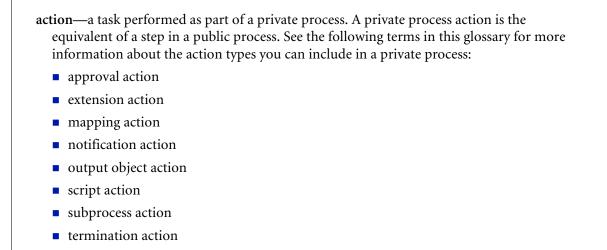Pentium is a registered trademark of Intel Corporation in the United States and/or other countries.

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 multiple implementations, and each implementation can have multiple 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 Partner Agreement Manager. 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